Note:

Before using this information and the product it supports read the information in “Notices” on page D-1.
## Contents

### Introduction
- About This Publication ... ix
  - Types of Users ... ix
  - Software Dependencies ... x
  - Assumptions About Your Locale ... x
  - Demonstration Database ... x
- New Features in Dynamic Server, Version 11.1 ... xi
- Documentation Conventions ... xiii
  - Typographical Conventions ... xiii
  - Feature, Product, and Platform Markup ... xiv
  - Example Code Conventions ... xiv
- Additional Documentation ... xv
  - Compliance with Industry Standards ... xv
  - Syntax Diagrams ... xv
  - How to Read a Command-Line Syntax Diagram ... xvi
  - Keywords and Punctuation ... xvii
  - Identifiers and Names ... xviii
- IBM Welcomes Your Comments ... xviii

### Chapter 1. System Catalog Tables
- In This Chapter ... 1-2
- Objects That the System Catalog Tables Track ... 1-2
- Using the System Catalog ... 1-2
- Structure of the System Catalog ... 1-8
  - SYSAGGREGATES (IDS) ... 1-10
  - SYSAUTH (IDS) ... 1-11
  - SYSATTRTYPES (IDS) ... 1-13
  - SYSBLOBS ... 1-13
  - SYSCASTS (IDS) ... 1-14
  - SYSCHECKS ... 1-15
  - SYSCHECKUDRDEP (IDS) ... 1-15
  - SYSCOLATTRIBS (IDS) ... 1-15
  - SYSCOLAUTH ... 1-16
  - SYSCOLDEPEND ... 1-17
  - SYSCOLUMNS ... 1-18
  - SYSCONSTRAINTS ... 1-21
  - SYSDEFAULTS ... 1-22
  - SYSDIR ... 1-23
  - SYSDIRECTIVES (IDS) ... 1-23
  - SYSDISTRIB ... 1-24
  - SYSDOMAINS (IDS) ... 1-25
  - SYSSERVICES (IDS) ... 1-25
  - SYSEXTCOLS (XPS) ... 1-26
  - SYSEXTCOLDFILES (XPS) ... 1-26
  - SYSEXTERNAL (XPS) ... 1-27
  - SYSFRAUTH (IDS) ... 1-27
  - SYSFRAUGHT ... 1-28
  - SYSINDEXES ... 1-30
  - SYSINCC (IDS) ... 1-31
  - SYSMULTIPLEX (IDS) ... 1-32
  - SYSSIZE (IDS) ... 1-32
  - SYSPRINTMAP (IDS) ... 1-33
  - SYSNEWDEPEND (XPS) ... 1-33
  - SYSOBJECTSTATE (IDS) ... 1-33
  - SYSPCCLASS (IDS) ... 1-34
Chapter 2. Data Types

In This Chapter .................................................................................................................. 2-2
Summary of Data Types .................................................................................................... 2-2
Description of Data Types .................................................................................................. 2-3
   BIGINT (XPS) .................................................................................................................. 2-5
   BLOB (IDS) .................................................................................................................... 2-5
   BOOLEAN (IDS) ............................................................................................................. 2-5
   BYTE ............................................................................................................................ 2-6
   CHAR(n) ....................................................................................................................... 2-7
   CHARACTER(n) ........................................................................................................... 2-8
   CHARACTER VARYING(m,r) ....................................................................................... 2-8
   CLOB (IDS) .................................................................................................................. 2-8
   DATE ........................................................................................................................... 2-9
   DATETIME ................................................................................................................... 2-10
   DEC ............................................................................................................................ 2-12
   DECIMAL .................................................................................................................... 2-12
   DISTINCT (IDS) ......................................................................................................... 2-14
   DOUBLE PRECISION ................................................................................................. 2-15
   FLOAT(n) ...................................................................................................................... 2-15
   IDSSECURITYLABEL ................................................................................................. 2-15
Chapter 3. Environment Variables

In This Chapter

Types of Environment Variables

Using Environment Variables on UNIX

Where to Set Environment Variables on UNIX

Setting Environment Variables in a Configuration File

Setting Environment Variables at Login Time

Syntax for Setting Environment Variables

Unsetting Environment Variables

Modifying an Environment-Variable Setting

Viewing Your Environment-Variable Settings

Checking Environment Variables with the chenv Utility

Rules of Precedence

Using Environment Variables on Windows

Where to Set Environment Variables on Windows

Environment Settings

Rules of Precedence

List of Environment Variables

Environment Variables
<table>
<thead>
<tr>
<th>Variable</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBACCNOIGN</td>
<td>3-16</td>
</tr>
<tr>
<td>DBANSIWARN</td>
<td>3-17</td>
</tr>
<tr>
<td>DBBLOBBUF</td>
<td>3-17</td>
</tr>
<tr>
<td>DBCENTURY</td>
<td>3-18</td>
</tr>
<tr>
<td>DBDATE</td>
<td>3-20</td>
</tr>
<tr>
<td>DBDELIMITER</td>
<td>3-22</td>
</tr>
<tr>
<td>DBEDIT</td>
<td>3-22</td>
</tr>
<tr>
<td>DBFLITMASK</td>
<td>3-23</td>
</tr>
<tr>
<td>DLALG</td>
<td>3-23</td>
</tr>
<tr>
<td>DBMONEY</td>
<td>3-24</td>
</tr>
<tr>
<td>DBNLS (IDS)</td>
<td>3-25</td>
</tr>
<tr>
<td>DBONLOAD (IDS)</td>
<td>3-26</td>
</tr>
<tr>
<td>DMBPATH</td>
<td>3-26</td>
</tr>
<tr>
<td>DBPRINT</td>
<td>3-28</td>
</tr>
<tr>
<td>DBREMTCMD (UNIX)</td>
<td>3-28</td>
</tr>
<tr>
<td>DBSPACETEMP</td>
<td>3-29</td>
</tr>
<tr>
<td>DBTEMP (IDS)</td>
<td>3-30</td>
</tr>
<tr>
<td>DBTIME</td>
<td>3-31</td>
</tr>
<tr>
<td>DBUPSPACE</td>
<td>3-33</td>
</tr>
<tr>
<td>DEFAULT_ATTACH</td>
<td>3-34</td>
</tr>
<tr>
<td>DELIMIDENT</td>
<td>3-34</td>
</tr>
<tr>
<td>ENVIGNORE (UNIX)</td>
<td>3-35</td>
</tr>
<tr>
<td>FET_BUF_SIZE</td>
<td>3-36</td>
</tr>
<tr>
<td>GLOBAL_DETACH_INFORM (XPS)</td>
<td>3-36</td>
</tr>
<tr>
<td>IBM_XPS_PARAMS (XPS)</td>
<td>3-37</td>
</tr>
<tr>
<td>IFMX_CART_ALRM (XPS)</td>
<td>3-38</td>
</tr>
<tr>
<td>IFMX_HISTORY_SIZE (XPS)</td>
<td>3-38</td>
</tr>
<tr>
<td>IFMX_OPT_FACT_TABS (XPS)</td>
<td>3-38</td>
</tr>
<tr>
<td>IFMX_OPT_NON_DIM_TABS (XPS)</td>
<td>3-39</td>
</tr>
<tr>
<td>IFX_DEF_TABLE_LOCKMODE (IDS)</td>
<td>3-39</td>
</tr>
<tr>
<td>IFX_DIRECTIVES</td>
<td>3-40</td>
</tr>
<tr>
<td>IFX_EXTDIRECTIVES</td>
<td>3-41</td>
</tr>
<tr>
<td>IFX_LONGID</td>
<td>3-41</td>
</tr>
<tr>
<td>IFX_NETBUF_PVTPPOOL_SIZE (UNIX)</td>
<td>3-42</td>
</tr>
<tr>
<td>IFX_NETBUF_SIZE</td>
<td>3-42</td>
</tr>
<tr>
<td>IFX_NO_LRU_TUNING</td>
<td>3-43</td>
</tr>
<tr>
<td>IFX_NO_TIMELIMIT_WARNING</td>
<td>3-43</td>
</tr>
<tr>
<td>IFX_NODBPROC</td>
<td>3-43</td>
</tr>
<tr>
<td>IFX_ONTAPE_FILE_PREFIX</td>
<td>3-44</td>
</tr>
<tr>
<td>IFX_PAD_VARCHAR (IDS)</td>
<td>3-44</td>
</tr>
<tr>
<td>IFX_UPDDESC (IDS)</td>
<td>3-44</td>
</tr>
<tr>
<td>IFX_XASTDCOMPLIANCE_XAEND</td>
<td>3-45</td>
</tr>
<tr>
<td>IFX_XFER_SHMBASE</td>
<td>3-45</td>
</tr>
<tr>
<td>IMADMIN</td>
<td>3-45</td>
</tr>
<tr>
<td>IMCCONFIG</td>
<td>3-46</td>
</tr>
<tr>
<td>IMCSERVER</td>
<td>3-46</td>
</tr>
<tr>
<td>INFORMIXC (UNIX)</td>
<td>3-47</td>
</tr>
<tr>
<td>INFORMIXCOCNSMCFG (IDS)</td>
<td>3-47</td>
</tr>
<tr>
<td>INFORMIXCONRETRY</td>
<td>3-47</td>
</tr>
<tr>
<td>INFORMIXCONRETRY</td>
<td>3-47</td>
</tr>
<tr>
<td>INFORMIXCONRETRY</td>
<td>3-47</td>
</tr>
<tr>
<td>INFORMIXCONTIME</td>
<td>3-48</td>
</tr>
<tr>
<td>INFORMIXCPPMAP (IDS)</td>
<td>3-49</td>
</tr>
<tr>
<td>INFORMIXDIR</td>
<td>3-49</td>
</tr>
<tr>
<td>INFORMIXKEYTAB (UNIX)</td>
<td>3-49</td>
</tr>
<tr>
<td>INFORMIXOCACHE (IDS)</td>
<td>3-50</td>
</tr>
<tr>
<td>INFORMIXSERVER</td>
<td>3-50</td>
</tr>
<tr>
<td>INFORMIXSHMBASE (UNIX)</td>
<td>3-51</td>
</tr>
<tr>
<td>INFORMIXSQLHOSTS</td>
<td>3-51</td>
</tr>
<tr>
<td>INFORMIXSTACKSIZE</td>
<td>3-52</td>
</tr>
<tr>
<td>INFORMIXTERM (UNIX)</td>
<td>3-52</td>
</tr>
<tr>
<td>INF_ROLE_SEP (IDS)</td>
<td>3-53</td>
</tr>
<tr>
<td>INTERACTIVE_DESKTOP_OFF (Windows)</td>
<td>3-54</td>
</tr>
</tbody>
</table>
Appendix A. The stores_demo Database. .............................................. A-1

Appendix B. The sales_demo and superstores_demo Databases ................. B-1

Appendix C. Accessibility ................................................................. C-1
Accessibility features for IBM Informix Dynamic Server .......................... C-1
   Accessibility Features. .................................................................... C-1
   Keyboard Navigation. ...................................................................... C-1
   Related Accessibility Information. ..................................................... C-1
   IBM and Accessibility .................................................................... C-1
Dotted Decimal Syntax Diagrams .......................................................... C-1

Notices ............................................................................................... D-1
Trademarks ...................................................................................... D-3

Index ................................................................................................. X-1
Introduction

About This Publication  ix
Types of Users  ix
Software Dependencies  x
Assumptions About Your Locale  x
Demonstration Database  x
New Features in Dynamic Server, Version 11.1  xi
Documentation Conventions  xiii
  Typographical Conventions  xiii
  Feature, Product, and Platform Markup  xiv
  Example Code Conventions  xiv
Additional Documentation  xv
Compliance with Industry Standards  xv
Syntax Diagrams  xv
  How to Read a Command-Line Syntax Diagram  xvi
  Keywords and Punctuation  xvii
  Identifiers and Names  xviii
IBM Welcomes Your Comments  xviii

About This Publication

This publication includes information about the system catalog tables, data types, and environment variables that IBM Informix products use.

This publication is one of a series of publications that discusses the Informix® implementation of SQL. The IBM Informix Guide to SQL: Syntax contains all the syntax descriptions for SQL and stored procedure language (SPL). The IBM Informix Guide to SQL: Tutorial shows how to use basic and advanced SQL and SPL routines to access and manipulate the data in your databases. The IBM Informix Database Design and Implementation Guide shows how to use SQL to implement and manage your databases.

See the documentation notes files for a list of the publications in the documentation set of your Informix database server.

Types of Users

This publication is written for the following users:

- Database users
- Database administrators
- Database server administrators
- Database-application programmers
- Performance engineers

This publication assumes that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with computer programming
• Some experience with database server administration, operating-system administration, or network administration

If you have limited experience with relational databases, SQL, or your operating system, refer to the IBM Informix Getting Started Guide for your database server for a list of supplementary titles.

Software Dependencies
This publication is written with the assumption that you are using one of the following database servers:
• IBM Informix Dynamic Server, Version 11.1
• IBM Informix Extended Parallel Server, Version 8.50

Assumptions About Your Locale
IBM Informix products can support many languages, cultures, and code sets. All the information related to character set, collation, and representation of numeric data, currency, date, and time is brought together in a single environment, called a Global Language Support (GLS) locale.

This publication assumes that your database uses the default locale. This default is en_us.8859-1 (ISO 8859-1) on UNIX® platforms or en_us.1252 (Microsoft® 1252) in Windows environments. This locale supports U.S. English format conventions for displaying and entering date, time, number, and currency values. It also supports the ISO 8859-1 (on UNIX and Linux®) or Microsoft 1252 (on Windows) code set, which includes the ASCII code set plus many 8-bit characters such as é, è, and ñ.

If you plan to use nondefault characters in your data or in SQL identifiers, or if you plan to use other collation rules for sorting character data, you need to specify the appropriate nondefault locale.

For instructions on how to specify a nondefault locale, and for additional syntax and other considerations related to GLS locales, see the IBM Informix GLS User’s Guide.

Demonstration Database
The DB–Access utility, which is provided with the database server products, includes one or more of the following demonstration databases:
• The stores_demo database illustrates a relational schema with information about a fictitious wholesale sporting-goods distributor. Many examples in IBM Informix publications are based on the stores_demo database.

Extended Parallel Server
• The sales_demo database illustrates a dimensional schema for data-warehousing applications. For conceptual information about dimensional data modeling, see the IBM Informix Database Design and Implementation Guide.

End of Extended Parallel Server

Dynamic Server
• The superstores_demo database illustrates an object-relational schema. The superstores_demo database contains examples of extended data types, type and
For information about how to create and populate the demonstration databases, see the IBM Informix DB–Access User’s Guide. For descriptions of the databases and their contents, see Appendix A, “The stores_demo Database,” on page A-1 and Appendix B, “The sales_demo and superstores_demo Databases,” on page B-1.

The scripts that you use to install the demonstration databases reside in the $INFORMIXDIR/bin directory on UNIX platforms and in the %INFORMIXDIR%in directory in Windows environments.

New Features in Dynamic Server, Version 11.1

For a comprehensive list of new database server features in Version 11.1, see the IBM Informix Getting Started Guide. In addition to documenting the new features that are described in this section, this manual corrects errata that have been identified since the previous edition.

- Enhanced Concurrency with Committed Read Isolation

In Committed Read isolation level, exclusive row-level locks held by other sessions can cause SQL operations to fail when attempting to read data in the locked rows. This release introduces a new LAST COMMITTED keyword option to the SET ISOLATION COMMITTED READ statement to reduce the risk of locking conflicts when attempting to read a table. This new syntax instructs Dynamic Server to return the most recently committed version of the rows, even if another concurrent session holds an exclusive row-level lock.

This behavior can be extended to the Dirty Read, Read Uncommitted, and Read Committed isolation levels by setting the new USELASTCOMMITTED configuration parameter or through new options to the SET ENVIRONMENT statement.

The IFX_DEF_TABLE_LOCKMODE environment variable can specify the default lock mode for database tables that are subsequently created without explicitly specifying the LOCKMODE PAGE or LOCKMODE ROW keywords. This feature is convenient if you need to create several tables of the same lock mode.

This feature supports B-tree indexes and functional indexes, but not R-tree indexes. It does not support tables that are being accessed by DataBlade modules, tables with columns of collection data types, tables created using a Virtual Table Interface, tables with page-level locking, tables with exclusive table-level locks, unlogged tables, or tables in databases with no transaction logging. For more information, see “IFX_DEF_TABLE_LOCKMODE (IDS)” on page 3-39.

- Enhanced Data-Type and UDR Support in Cross-Server Distributed Operations

Earlier releases of Informix Dynamic Server restricted the remote execution of UDRs in databases of other Dynamic Server instances to SPL routines that the EXECUTE FUNCTION or EXECUTE PROCEDURE statement invoke explicitly, and to SPL routines that queries and other DML operations invoked implicitly. This release extends support for UDRs in cross-database and cross-server distributed operations to most contexts where a UDR is valid in the local database. In addition, external routines written in the C or Java languages are now valid in any distributed operation where an SPL routine is valid.
This release also extends the data types that are valid as parameters or return values of cross-server UDRs, which were formerly restricted to non-opaque built-in SQL data types, by supporting these additional data types:

- BOOLEAN
- LVARCHAR
- DISTINCT of built-in types that are not opaque
- DISTINCT of BOOLEAN
- DISTINCT of LVARCHAR
- DISTINCT of the DISTINCT types listed above.

These data types can be returned by SPL, C, or Java language UDRs that use these data types as parameters or as return values, if the UDRs are defined in all the participating databases. Any implicit or explicit casts defined over these data types must be duplicated across all the participating Dynamic Server instances. The DISTINCT data types must have exactly the same data type hierarchy defined in all databases that participate in the distributed query.

This feature does not relax existing restrictions on other opaque and DISTINCT types or on large-object, serial, and collection data types in locally or remotely executed SPL routines or external routines. For more information about this feature, see “IFX_PAD_VARCHAR (IDS) on page 3-44”.

- Trigger Enhancements

New features expand the syntax and the functionality of triggers on tables and on views.

- You can now define multiple triggers for INSERT, DELETE, UPDATE, and SELECT types of triggering event on the same table, and multiple INSTEAD OF triggers for INSERT, DELETE, and UPDATE types of triggering event on the same view.

- New syntax features of the CREATE FUNCTION and CREATE PROCEDURE statements can create SPL trigger routines that the FOR EACH ROW clause of the CREATE TRIGGER statement can invoke as a triggered action. A trigger routine can execute procedural SPL statements, including LET statements that reference NEW and OLD correlation names to access or assign new values to a column in a triggered action.

These features make it easier to incorporate Dynamic Server triggers on tables and views within a heterogeneous information management system where multiple applications need to update the same table or view. For more information about these trigger features, see “SYSTRIGGERS” on page 1-51.

- Session Configuration Procedures

This feature provides a mechanism to enable users to change the properties of their session environment at connection or access time without changing the application that the session runs. This feature is useful if the user cannot modify the source code of the application to include session-related SQL statements, as is typically the case with third-party applications.

This feature is implemented by two SPL procedures, called sysdbopen() and sysdbclose(), which the DBA or user informix can define. Each database can include versions of these routines for different users and for PUBLIC.

- The sysdbopen() procedure, if it exists in the database, is invoked at connection time when the CONNECT or DATABASE statement executes successfully. This routine can include SQL statements that specify the isolation level, the current role, and session environment attributes.

- If user.sysdbopen() exists, this procedure executes automatically when that user connects to the database.
- If PUBLIC.sysdbopen() exists, the PUBLIC.sysdbopen() procedure executes automatically when a user who is the owner of no user.sysdbopen() procedure connects to the database.

- The sysdbclose() procedure, if it exists in the database, is invoked when the DISCONNECT or CLOSE DATABASE statement terminates the connection to the database. This routine can include SQL statements to accomplish tasks like dropping temporary table, calculating statistics, or resetting the current role after disconnecting from the database.

- If user.sysdbclose() exists, this procedure executes automatically when that user disconnects from the database.

- If PUBLIC.sysdbclose() exists, the PUBLIC.sysdbclose() procedure executes automatically when a user who is the owner of no user.sysdbclose() procedure disconnects from the database.

The DBA privilege is required to create, drop, or alter these procedures. In databases that are not ANSI-compliant, the sysdbopen() and sysdbclose() routines are exceptions to the rule that the owner name is not part of the routine signature of a UDR.

These SPL routines cannot be written in an external language, and they cannot be defined with arguments or with return values. For more information about these session configuration procedures, see "Where to Set Environment Variables on UNIX" on page 3-4.

Label Based Access Control

Label-based access control (LBAC) is an implementation of multi-level security (MLS) that enables you to control who has read access and who has write access to individual rows and columns of data. The access is controlled through the following catalog tables:

- "SYSSECLABELCOMPONENTS" on page 1-42
- "SYSSECLABELCOMPONENTELEMENTS" on page 1-43
- "SYSSECPOLICIES" on page 1-43
- "SYSSECPOLICYCOMPONENTS" on page 1-44
- "SYSSECPOLICYEXEMPTIONS" on page 1-44
- "SYSSECLABELS" on page 1-44
- "SYSSECLABELNAMES" on page 1-45
- "SYSSECLABELAUTH" on page 1-45
- "SYSSECLABELCOMPONENTS" on page 1-42

Documentation Conventions

This section describes the following conventions, which are used in the product documentation for IBM Informix Dynamic Server:

- Typographical conventions
- Feature, product, and platform conventions
- Syntax diagrams
- Command-line conventions
- Example code conventions

Typographical Conventions

This publication uses the following conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.
<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEYWORD</td>
<td>Keywords of SQL, SPL, and some other programming languages appear in uppercase letters in a serif font.</td>
</tr>
<tr>
<td><em>italics</em></td>
<td>Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.</td>
</tr>
<tr>
<td><strong>boldface</strong></td>
<td>Names of program entities (such as classes, events, and tables), environment variables, file names, path names, and interface elements (such as icons, menu items, and buttons) appear in boldface.</td>
</tr>
<tr>
<td><em>monospace</em></td>
<td>Information that the product displays and information that you enter appear in a monospace typeface.</td>
</tr>
<tr>
<td><strong>KEYSTROKE</strong></td>
<td>Keys that you are to press appear in uppercase letters in a sans serif font.</td>
</tr>
<tr>
<td>&gt;</td>
<td>This symbol indicates a menu item. For example, “Choose <strong>Tools &gt; Options</strong>” means choose the <strong>Options</strong> item from the <strong>Tools</strong> menu.</td>
</tr>
</tbody>
</table>

**Feature, Product, and Platform Markup**

Feature, product, and platform markup identifies paragraphs that contain feature-specific, product-specific, or platform-specific information. Some examples of this markup follow:

---
**Dynamic Server**

Identifies information that is specific to IBM Informix Dynamic Server

---
**End of Dynamic Server**

---
**Windows Only**

Identifies information that is specific to the Windows operating system

---
**End of Windows Only**

This markup can apply to one or more paragraphs within a section. When an entire section applies to a particular product or platform, this is noted as part of the heading text, for example:

**Table Sorting (Windows)**

**Example Code Conventions**

Examples of SQL code occur throughout this publication. Except as noted, the code is not specific to any single IBM Informix application development tool.

If only SQL statements are listed in the example, they are not delimited by semicolons. For instance, you might see the code in the following example:

```sql
CONNECT TO stores_demo
...
DELETE FROM customer
   WHERE customer_num = 121
...
COMMIT WORK
DISCONNECT CURRENT
```
To use this SQL code for a specific product, you must apply the syntax rules for that product. For example, if you are using DB–Access, you must delimit multiple statements with semicolons. If you are using an SQL API, you must use EXEC SQL at the start of each statement and a semicolon (or other appropriate delimiter) at the end of the statement.

Tip: Ellipsis points in a code example indicate that more code would be added in a full application, but it is not necessary to show it to describe the concept being discussed.

For detailed directions on using SQL statements for a particular application development tool or SQL API, see the documentation for your product.

Additional Documentation

You can view, search, and print all of the product documentation from the IBM Informix Dynamic Server information center on the Web at http://publib.boulder.ibm.com/infocenter/idshelp/v111/index.jsp.

For additional documentation about IBM Informix Dynamic Server and related products, including release notes, machine notes, and documentation notes, go to the online product library page at http://www.ibm.com/software/data/informix/pubs/library/ Alternatively, you can access or install the product documentation from the Quick Start CD that is shipped with the product.

Compliance with Industry Standards

The American National Standards Institute (ANSI) and the International Organization of Standardization (ISO) have jointly established a set of industry standards for the Structured Query Language (SQL). IBM® Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992. In addition, many features of IBM Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL Common Applications Environment (CAE) standards.

Syntax Diagrams

This guide uses syntax diagrams built with the following components to describe the syntax for statements and all commands other than system-level commands.

<table>
<thead>
<tr>
<th>Component represented in PDF</th>
<th>Component represented in HTML</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------------</td>
<td>&gt;&gt;-----------------------------</td>
<td>Statement begins.</td>
</tr>
<tr>
<td>-------------------------</td>
<td>------------------------------</td>
<td>Statement continues on next line.</td>
</tr>
<tr>
<td>---------------</td>
<td>&gt;-----------------------------</td>
<td>Statement continues from previous line.</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------</td>
<td>Statement ends.</td>
</tr>
<tr>
<td>SELECT -----</td>
<td>-------SELECT--------------</td>
<td>Required item.</td>
</tr>
</tbody>
</table>
Table 1. Syntax Diagram Components (continued)

<table>
<thead>
<tr>
<th>Component represented in PDF</th>
<th>Component represented in HTML</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOCAL</td>
<td>'------LOCAL------'</td>
<td>Optional item.</td>
</tr>
<tr>
<td>ALL</td>
<td>'------ALL------'</td>
<td>Required item with choice. One and only one item must be present.</td>
</tr>
<tr>
<td>DISTINCT</td>
<td>'------DISTINCT------'</td>
<td></td>
</tr>
<tr>
<td>UNIQUE</td>
<td>'------UNIQUE------'</td>
<td></td>
</tr>
<tr>
<td>FOR UPDATE</td>
<td>'------FOR UPDATE------'</td>
<td>Optional items with choice are shown below the main line, one of which you might specify.</td>
</tr>
<tr>
<td>FOR READ ONLY</td>
<td>'------FOR READ ONLY------'</td>
<td></td>
</tr>
<tr>
<td>NEXT</td>
<td>'------NEXT------'</td>
<td>The values below the main line are optional, one of which you might specify. If you do not specify an item, the value above the line will be used as the default.</td>
</tr>
<tr>
<td>PRIOR</td>
<td>'------PRIOR------'</td>
<td></td>
</tr>
<tr>
<td>PREVIOUS</td>
<td>'------PREVIOUS------'</td>
<td></td>
</tr>
<tr>
<td>index_name</td>
<td>'------index_name------'</td>
<td>Optional items. Several items are allowed; a comma must precede each repetition.</td>
</tr>
<tr>
<td>table_name</td>
<td>'------table_name------'</td>
<td></td>
</tr>
</tbody>
</table>

How to Read a Command-Line Syntax Diagram

The following command-line syntax diagram uses some of the elements listed in the table in Syntax Diagrams.

**Creating a No-Conversion Job**

```
onpladm create job -j job -p project -n d -D database -t table
```

```
npladm create job -j job -p project -n d -D database -t table
```
Notes:

1. See page Z-1

The second line in this diagram has a segment named “Setting the Run Mode,” which according to the diagram footnote, is on page Z-1. If this was an actual cross-reference, you would find this segment in on the first page of Appendix Z. Instead, this segment is shown in the following segment diagram. Notice that the diagram uses segment start and end components.

**Setting the Run Mode:**

To see how to construct a command correctly, start at the top left of the main diagram. Follow the diagram to the right, including the elements that you want. The elements in this diagram are case sensitive because they illustrate utility syntax. Other types of syntax, such as SQL, are not case sensitive.

The Creating a No-Conversation Job diagram illustrates the following steps:

1. Type `onpladm create job` and then the name of the job.
2. Optionally, type `-p` and then the name of the project.
3. Type the following required elements:
   * `-n`
   * `-d` and the name of the device
   * `-D` and the name of the database
   * `-t` and the name of the table
4. Optionally, you can choose one or more of the following elements and repeat them an arbitrary number of times:
   * `-S` and the server name
   * `-T` and the target server name
   * The run mode. To set the run mode, follow the Setting the Run Mode segment diagram to type `-f`, optionally type `d`, `p`, or `a`, and then optionally type `l` or `u`.
5. Follow the diagram to the terminator.

**Keywords and Punctuation**

Keywords are words reserved for statements and all commands except system-level commands. When a keyword appears in a syntax diagram, it is
shown in uppercase letters. When you use a keyword in a command, you can write it in uppercase or lowercase letters, but you must spell the keyword exactly as it appears in the syntax diagram.

You must also use any punctuation in your statements and commands exactly as shown in the syntax diagrams.

**Identifiers and Names**

Variables serve as placeholders for identifiers and names in the syntax diagrams and examples. You can replace a variable with an arbitrary name, identifier, or literal, depending on the context. Variables are also used to represent complex syntax elements that are expanded in additional syntax diagrams. When a variable appears in a syntax diagram, an example, or text, it is shown in *lowercase* italic.

The following syntax diagram uses variables to illustrate the general form of a simple SELECT statement.

```
>>> SELECT—column_name—FROM—table_name
```

When you write a SELECT statement of this form, you replace the variables `column_name` and `table_name` with the name of a specific column and table.

---

**IBM Welcomes Your Comments**

We want to know about any corrections or clarifications that you would find useful in our publications, which will help us improve future versions. Include the following information:

- The name and version of the publication that you are using
- Section and page number
- Your suggestions about the publication

Send your comments to us at the following e-mail address:

[docinf@us.ibm.com](mailto:docinf@us.ibm.com)

This e-mail address is reserved for reporting errors and omissions in our documentation. For immediate help with a technical problem, contact IBM Technical Support. For instructions, see the IBM Informix Technical Support website at [http://www.ibm.com/planetwide/](http://www.ibm.com/planetwide/)

We appreciate your suggestions.
Chapter 1. System Catalog Tables

In This Chapter ................................................. 1-2
Objects That the System Catalog Tables Track .................. 1-2
Using the System Catalog ...................................... 1-2
Structure of the System Catalog ................................ 1-8
SYSAGGREGATES (IDS) ........................................ 1-10
SYSAMS (IDS) .................................................. 1-11
SYSATRTYPES (IDS) .......................................... 1-13
SYSBLOBS ..................................................... 1-13
SYSCASTS (IDS) .............................................. 1-14
SYSCHECKS ................................................... 1-15
SYSCHECKUDRDEP (IDS) .................................... 1-15
SYSCOLATTRIBS (IDS) ..................................... 1-15
SYSCOLAUTH ................................................ 1-16
SYSCOLDEPEND .............................................. 1-17
SYSCOLUMNS ................................................ 1-18
SYSCONSTRAINTS ............................................ 1-21
SYSDEFAULTS ................................................. 1-22
SYSDEPEND .................................................. 1-23
SYSDIRECTIVES (IDS) ....................................... 1-23
SYSDISTRIB .................................................. 1-24
SYSDOMAINS (IDS) .......................................... 1-25
SYSErrORS (IDS) ............................................ 1-25
SYSEXTCOLS (XPS) ......................................... 1-26
SYSEXTCOLs (XPS) ......................................... 1-26
SYSEXTERNAL (XPS) ........................................ 1-27
SYSFRAGAUTH (IDS) ........................................ 1-27
SYSFRAGMENTS ............................................. 1-28
SYSINDEXES ............................................... 1-30
SYSINDICES (IDS) .......................................... 1-31
SYSINHERITS (IDS) ......................................... 1-32
SYSLANGAUTH (IDS) ....................................... 1-32
SYSLOGMAP (IDS) .......................................... 1-33
SYSNEWDEPEND (XPS) ..................................... 1-33
SYSOBJSTATE (IDS) ........................................ 1-33
SYSOPCLASSES (IDS) ...................................... 1-34
SYSOPCLSTR ............................................... 1-35
SYSPROCAUTH .............................................. 1-36
SYSPROCBODY .............................................. 1-36
SYSPROCColumns ............................................ 1-36
SYSPROCEDURES .......................................... 1-37
SYSPROCPLAN .............................................. 1-37
SYSREFERENCES ............................................ 1-40
SYSREPOSITORY (XPS) ...................................... 1-40
SYSROLEAUTH .............................................. 1-41
SYSROUTINELANGS (IDS) .................................. 1-41
SYSSECLABELCOMPONENTS .................................. 1-42
SYSSECLABELCOMPONENTELEMENTS ......................... 1-42
SYSSECPOlicIES .......................................... 1-43
SYSSECPOlicYCOMPONENTS ................................ 1-43
SYSSECPOlicYEXEMPTIONS ................................ 1-44
SYSSECLABELS ............................................. 1-44
SYSSECLABELNAMES ....................................... 1-45
SYSSECLABELAUTH ......................................... 1-45
SYSSEQUENCES (IDS) ....................................... 1-45
SYSSYNONYMS .............................................. 1-45
In This Chapter

The system catalog consists of tables that describe the structure of the database. Sometimes called the data dictionary, these tables contain everything that the database knows about itself. Each system catalog table contains specific information about elements in the database.

This chapter provides information about the structure, content, and use of the system catalog tables. It also discusses the Information Schema, which provides information about the tables, views, and columns on the current database server.

Objects That the System Catalog Tables Track

The system catalog tables maintain information about the database, including the following categories of database objects:

- Tables, views, and synonyms
- Columns, constraints, indexes, and fragments
- Triggers
- Procedures, functions, routines, and associated messages
- Authorized users, roles, and privileges to access database objects
- Data types and casts (IDS)
- Aggregate functions (IDS)
- Access methods and operator classes (IDS)
- Sequence objects (IDS)
- External optimizer directives (IDS)
- Inheritance relationships (IDS)

Using the System Catalog

Informix database servers automatically generate the system catalog tables when you create a database. You can query the system catalog tables as you would query any other table in the database. The system catalog tables for a newly created database reside in a common area of the disk called a dbspace. Every database has
its own system catalog tables. All tables and views in the system catalog have the
prefix sys (for example, the systables system catalog table).

Not all tables with the prefix sys are true system catalog tables. For example, the
syscdr database supports the Enterprise Replication feature. Non-catalog tables,
however, have a tabid >= 100. System catalog tables all have a tabid < 100. See
later in this section and ["SYSTABLES" on page 1-48] for more information about
tabid numbers that the database server assigns to tables, views, synonyms, and (in
Dynamic Server) sequence objects.

Tip: Do not confuse the system catalog tables of a database with the tables in the
sysmaster, sysutils, syscdr, or (for Dynamic Server) the sysuser databases.
The names of tables in those databases also have the sys prefix, but they
contain information about an entire database server, which might manage
multiple databases. Information in the sysmaster, sysutils, syscdr, and
sysuser tables is primarily useful for database server administrators (DBSAs).
See also the IBM Informix Administrator’s Guide and IBM Informix
Administrator’s Reference.

The database server accesses the system catalog constantly. Each time an SQL
statement is processed, the database server accesses the system catalog to
determine system privileges, add or verify table or column names, and so on.

For example, the following CREATE SCHEMA block adds the customer table, with
its indexes and privileges, to the stores_demo database. This block also adds a
view, california, which restricts the data of the customer table to only the first and
last names of the customer, the company name, and the telephone number for all
customers who reside in California.

CREATE SCHEMA AUTHORIZATION mary
CREATE TABLE customer (customer_num SERIAL(101), fname CHAR(15),
   lname CHAR(15), company CHAR(20), address1 CHAR(20), address2 CHAR(20),
   city CHAR(15), state CHAR(2), zipcode CHAR(5), phone CHAR(18))
GRANT ALTER, ALL ON customer TO cathl WITH GRANT OPTION AS mary
GRANT SELECT ON customer TO public
GRANT UPDATE (fname, lname, phone) ON customer TO nhowe
CREATE VIEW california AS
   SELECT fname, lname, company, phone FROM customer WHERE state = 'CA'
CREATE UNIQUE INDEX c_num_ix ON customer (customer_num)
CREATE INDEX state_ix ON customer (state)

To process this CREATE SCHEMA block, the database server first accesses the
system catalog to verify the following information:

• The new table and view names do not already exist in the database. (If the
database is ANSI-compliant, the database server verifies that the new names do
not already exist for the specified owners.)
• The user has permission to create tables and grant user privileges.
• The column names in the CREATE VIEW and CREATE INDEX statements exist
in the customer table.

In addition to verifying this information and creating two new tables, the database
server adds new rows to the following system catalog tables:

• systables
• syscolumns
• sysviews
• systabauth
• syscolauth
The following two new rows of information are added to the systables system catalog table after the CREATE SCHEMA block is run.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>First Row</th>
<th>Second Row</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabname</td>
<td>customer</td>
<td>california</td>
</tr>
<tr>
<td>owner</td>
<td>maryl</td>
<td>maryl</td>
</tr>
<tr>
<td>partnum</td>
<td>16778361</td>
<td>0</td>
</tr>
<tr>
<td>tabid</td>
<td>101</td>
<td>102</td>
</tr>
<tr>
<td>rowsize</td>
<td>134</td>
<td>134</td>
</tr>
<tr>
<td>ncols</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>nindexes</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>nrows</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>created</td>
<td>01/26/2007</td>
<td>01/26/2007</td>
</tr>
<tr>
<td>version</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>tabtype</td>
<td>T</td>
<td>V</td>
</tr>
<tr>
<td>locklevel</td>
<td>P</td>
<td>B</td>
</tr>
<tr>
<td>npused</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fextsize</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>nextsize</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>flags</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>site</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dbname</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each table recorded in the systables system catalog table is assigned a tabid, a system-assigned sequential number that uniquely identifies each table in the database. The system catalog tables receive 2-digit tabid numbers, and the user-created tables receive sequential tabid numbers that begin with 100.

The CREATE SCHEMA block adds 14 rows to the syscolumns system catalog table. These rows correspond to the columns in the table customer and the view california, as the following example shows.
In the **syscolumns** table, each column within a table is assigned a sequential column number, **colno**, that uniquely identifies the column within its table. In the **colno** column, the **fname** column of the **customer** table is assigned the value 2 and the **fname** column of the view **california** is assigned the value 1.

The **colmin** and **colmax** columns are empty. These columns contain values when a column is the first key (or the only key) in an index, has no NULL or duplicate values, and the UPDATE STATISTICS statement has been run.

The database server also adds rows to the **sysviews** system catalog table, whose **viewtext** column contains each line of the CREATE VIEW statement that defines the view. In that column, the **x0** that precedes the column names in the statement (for example, **x0.fname**) operates as an alias that distinguishes among the same columns that are used in a self-join.

The CREATE SCHEMA block also adds rows to the **systabauth** system catalog table. These rows correspond to the user privileges granted on **customer** and **california** tables, as the following example shows.

<table>
<thead>
<tr>
<th>grantor</th>
<th>grantee</th>
<th>tabid</th>
<th>tabauth</th>
</tr>
</thead>
<tbody>
<tr>
<td>maryl</td>
<td>public</td>
<td>101</td>
<td>su-idx--</td>
</tr>
<tr>
<td>maryl</td>
<td>cathl</td>
<td>101</td>
<td>SU-IDXAR</td>
</tr>
<tr>
<td>maryl</td>
<td>nhowe</td>
<td>101</td>
<td>--*-----</td>
</tr>
<tr>
<td></td>
<td>maryl</td>
<td>102</td>
<td>SU-ID---</td>
</tr>
</tbody>
</table>

The **tabauth** column specifies the table-level privileges granted to users on the **customer** and **california** tables. This column uses an 8-byte pattern, such as **s** (Select), **u** (Update), ***** (column-level privilege), **i** (Insert), **d** (Delete), **x** (Index), **a** (Alter), and **r** (References), to identify the type of privilege. In this example, the user **nhowe** has column-level privileges on the **customer** table. Where a hyphen ( - ) appears, the user has not been granted the privilege whose position the hyphen occupies within the **tabauth** value.
If the `tabauth` privilege code appears in uppercase (for example, `S` for Select), the user has this privilege and can also grant it to others; but if the privilege code is lowercase (for example, `s` for Select), the user cannot grant it to others.

In addition, three rows are added to the `syscolauth` system catalog table. These rows correspond to the user privileges that are granted on specific columns in the `customer` table as the following example shows.

<table>
<thead>
<tr>
<th>grantor</th>
<th>grante</th>
<th>tabid</th>
<th>colno</th>
<th>colauth</th>
</tr>
</thead>
<tbody>
<tr>
<td>maryl</td>
<td>nhowe</td>
<td>101</td>
<td>2</td>
<td>-u-</td>
</tr>
<tr>
<td>maryl</td>
<td>nhowe</td>
<td>101</td>
<td>3</td>
<td>-u-</td>
</tr>
<tr>
<td>maryl</td>
<td>nhowe</td>
<td>101</td>
<td>10</td>
<td>-u-</td>
</tr>
</tbody>
</table>

The `colauth` column specifies the column-level privileges that are granted on the `customer` table. This column uses a 3-byte, pattern such as `s` (Select), `u` (Update), and `r` (References), to identify the type of privilege. For example, the user `nhowe` has Update privileges on the second column (because the `colno` value is 2) of the `customer` table (indicated by `tabid` value of 101).

The `CREATE SCHEMA` block adds two rows to the `sysindexes` system catalog table (the `sysindices` table for Dynamic Server). These rows correspond to the indexes created on the `customer` table, as the following example shows.
In this table, the idxtype column identifies whether the created index requires unique values (U) or accepts duplicate values (D). For example, the c_num_ix index on the customer.customer_num column is unique.

### Accessing the System Catalog

Normal user access to the system catalog is read-only. Users with Connect or Resource privileges cannot alter the catalog, but they can access data in the system catalog tables on a read-only basis using standard SELECT statements.

For example, the following SELECT statement displays all the table names and corresponding tabid codes of user-created tables in the database:

```
SELECT tabname, tabid FROM systables WHERE tabid > 99
```

When you use DB–Access, only the tables that you created are displayed. To display the system catalog tables, enter the following statement:

```
SELECT tabname, tabid FROM systables WHERE tabid < 100
```

You can use the SUBSTR or SUBSTRING function to select only part of a source string. To display the list of tables in columns, enter the following statement:
SELECT SUBSTR(tabname, 1, 18), tabid FROM systables

Although user informix can modify most system catalog tables, it is strongly recommended that you do not update, delete, or insert any rows in them. Modifying system catalog tables can destroy the integrity of the database. The ALTER TABLE statement cannot modify the size of the next extent of system catalog tables.

For certain catalog tables of Dynamic Server, however, it is valid to add entries to the system catalog tables. For instance, in the case of the syserrors system catalog table and the systracemsgs system catalog table, a DataBlade module developer can directly insert entries that appear in these system catalog tables.

### Updating System Catalog Data

In Informix database servers, the optimizer determines the most efficient strategy for executing SQL queries. The optimizer allows you to query the database without having to consider fully which tables to search first in a join or which indexes to use. The optimizer uses information from the system catalog to determine the best query strategy.

If you use the UPDATE STATISTICS statement to update the system catalog before executing a query, you can ensure that the information provided to the optimizer is current. When you delete or modify a table, the database server does not automatically update the related statistical data in the system catalog. For example, if you delete one or more rows in a table with the DELETE statement, the nrows column in the systables system catalog table, which holds the number of rows for that table, is not updated automatically.

The UPDATE STATISTICS statement causes the database server to recalculate data in the systables, sysdistrib, syscolumns, and sysindexes (sysindices for Dynamic Server) system catalog tables. After you run UPDATE STATISTICS, the systables system catalog table holds the correct value in the nrows column. If you specify MEDIUM or HIGH mode when you run UPDATE STATISTICS, the sysdistrib system catalog table holds the updated data-distribution data.

Whenever you modify a data table extensively, use the UPDATE STATISTICS statement to update data in the system catalog. For more information on the UPDATE STATISTICS statement, see the IBM Informix Guide to SQL: Syntax.

### Structure of the System Catalog

The following system catalog tables describe the structure of an Informix database. Here X indicates whether IDS, XPS, or both support the table.

<table>
<thead>
<tr>
<th>System Catalog Table</th>
<th>XPS</th>
<th>IDS</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sysaggregates</td>
<td></td>
<td>X</td>
<td>1-10</td>
</tr>
<tr>
<td>sysams</td>
<td></td>
<td>X</td>
<td>1-11</td>
</tr>
<tr>
<td>sysattrtypes</td>
<td></td>
<td>X</td>
<td>1-12</td>
</tr>
<tr>
<td>sysblobs</td>
<td></td>
<td>X</td>
<td>1-13</td>
</tr>
<tr>
<td>syscasts</td>
<td></td>
<td>X</td>
<td>1-14</td>
</tr>
<tr>
<td>syschecks</td>
<td></td>
<td>X</td>
<td>1-15</td>
</tr>
<tr>
<td>syscheckudrdep</td>
<td></td>
<td>X</td>
<td>1-15</td>
</tr>
<tr>
<td>syscolattrs</td>
<td></td>
<td>X</td>
<td>1-15</td>
</tr>
<tr>
<td>System Catalog Table</td>
<td>XPS</td>
<td>IDS</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>syscolauth</td>
<td>X</td>
<td>X</td>
<td>1-16</td>
</tr>
<tr>
<td>syscoldepend</td>
<td>X</td>
<td>X</td>
<td>1-17</td>
</tr>
<tr>
<td>syscolumns</td>
<td>X</td>
<td>X</td>
<td>1-18</td>
</tr>
<tr>
<td>sysconstraints</td>
<td>X</td>
<td>X</td>
<td>1-21</td>
</tr>
<tr>
<td>sysdefaults</td>
<td>X</td>
<td>X</td>
<td>1-22</td>
</tr>
<tr>
<td>sysdepend</td>
<td>X</td>
<td>X</td>
<td>1-23</td>
</tr>
<tr>
<td>sysdirectives</td>
<td></td>
<td>X</td>
<td>1-23</td>
</tr>
<tr>
<td>sysdistrib</td>
<td>X</td>
<td>X</td>
<td>1-24</td>
</tr>
<tr>
<td>sysdomains</td>
<td>X</td>
<td></td>
<td>1-25</td>
</tr>
<tr>
<td>sysexcerrors</td>
<td></td>
<td>X</td>
<td>1-25</td>
</tr>
<tr>
<td>sysextcols</td>
<td>X</td>
<td></td>
<td>1-26</td>
</tr>
<tr>
<td>sysextdfiles</td>
<td>X</td>
<td></td>
<td>1-26</td>
</tr>
<tr>
<td>sysexternal</td>
<td>X</td>
<td></td>
<td>1-27</td>
</tr>
<tr>
<td>sysfragauth</td>
<td></td>
<td>X</td>
<td>1-27</td>
</tr>
<tr>
<td>sysfragments</td>
<td>X</td>
<td>X</td>
<td>1-28</td>
</tr>
<tr>
<td>sysindexes</td>
<td>X</td>
<td>X</td>
<td>1-30</td>
</tr>
<tr>
<td>sysindices</td>
<td>X</td>
<td></td>
<td>1-31</td>
</tr>
<tr>
<td>sysinherits</td>
<td>X</td>
<td></td>
<td>1-32</td>
</tr>
<tr>
<td>syslangauth</td>
<td></td>
<td>X</td>
<td>1-32</td>
</tr>
<tr>
<td>syslogmap</td>
<td>X</td>
<td></td>
<td>1-33</td>
</tr>
<tr>
<td>sysnewdepend</td>
<td>X</td>
<td></td>
<td>1-33</td>
</tr>
<tr>
<td>sysobjstate</td>
<td>X</td>
<td></td>
<td>1-33</td>
</tr>
<tr>
<td>sysopclasses</td>
<td>X</td>
<td></td>
<td>1-34</td>
</tr>
<tr>
<td>sysopclstr</td>
<td>X</td>
<td></td>
<td>1-35</td>
</tr>
<tr>
<td>sysproauth</td>
<td>X</td>
<td>X</td>
<td>1-36</td>
</tr>
<tr>
<td>sysproccolumns</td>
<td>X</td>
<td>X</td>
<td>1-37</td>
</tr>
<tr>
<td>sysprobody</td>
<td>X</td>
<td>X</td>
<td>1-36</td>
</tr>
<tr>
<td>sysprocedures</td>
<td>X</td>
<td>X</td>
<td>1-37</td>
</tr>
<tr>
<td>sysproplan</td>
<td>X</td>
<td>X</td>
<td>1-40</td>
</tr>
<tr>
<td>sysreferences</td>
<td>X</td>
<td>X</td>
<td>1-41</td>
</tr>
<tr>
<td>sysrepository</td>
<td>X</td>
<td></td>
<td>1-41</td>
</tr>
<tr>
<td>sysroleauth</td>
<td>X</td>
<td>X</td>
<td>1-42</td>
</tr>
<tr>
<td>sysroutinelangs</td>
<td>X</td>
<td></td>
<td>1-42</td>
</tr>
<tr>
<td>sysseclabelcomponents</td>
<td>X</td>
<td></td>
<td>1-42</td>
</tr>
<tr>
<td>sysseclabelcomponentelements</td>
<td>X</td>
<td>1-43</td>
<td></td>
</tr>
<tr>
<td>syssecpolicies</td>
<td>X</td>
<td></td>
<td>1-43</td>
</tr>
<tr>
<td>syssecpolicycomponents</td>
<td>X</td>
<td></td>
<td>1-44</td>
</tr>
<tr>
<td>syssecpolicyexemptions</td>
<td>X</td>
<td></td>
<td>1-44</td>
</tr>
<tr>
<td>sysseclabels</td>
<td>X</td>
<td></td>
<td>1-44</td>
</tr>
<tr>
<td>sysseclabelnames</td>
<td>X</td>
<td></td>
<td>1-45</td>
</tr>
<tr>
<td>sysseclabelauth</td>
<td>X</td>
<td></td>
<td>1-45</td>
</tr>
</tbody>
</table>
In the default database locale (U. S. English, ISO 8859-1 codeset), character column types in these tables are CHAR and VARCHAR. For all other locales, character column types are national character types, NCHAR and NVARCHAR. For information about collation order of data types, see the IBM Informix GLS User’s Guide. See also Chapter 2 of this publication.

SYSAGGREGATES (IDS)

The `sysaggregates` system catalog table records user-defined aggregates (UDAs). The `sysaggregates` table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>VARCHAR(128)</td>
<td>Name of the aggregate</td>
</tr>
<tr>
<td>owner</td>
<td>CHAR(32)</td>
<td>Name of the owner of the aggregate</td>
</tr>
<tr>
<td>aggid</td>
<td>SERIAL</td>
<td>Unique code identifying the aggregate</td>
</tr>
<tr>
<td>init_func</td>
<td>VARCHAR(128)</td>
<td>Name of initialization UDR</td>
</tr>
<tr>
<td>iter_func</td>
<td>VARCHAR(128)</td>
<td>Name of iterator UDR</td>
</tr>
<tr>
<td>combine_func</td>
<td>VARCHAR(128)</td>
<td>Name of combine UDR</td>
</tr>
<tr>
<td>final_func</td>
<td>VARCHAR(128)</td>
<td>Name of finalization UDR</td>
</tr>
<tr>
<td>handlesnulls</td>
<td>BOOLEAN</td>
<td>NULL-handling indicator: t = handles NULLs, f = does not handle NULLs</td>
</tr>
</tbody>
</table>

Each user-defined aggregate has one entry in `sysaggregates` that is uniquely identified by its identifying code (the `aggid` value). Only user-defined aggregates (aggregates that are not built in) have entries in `sysaggregates`. 
Both a simple index on the *aggid* column and a composite index on the *name* and *owner* columns require unique values.

**SYSAMS (IDS)**

The *sysams* system catalog table contains information that is needed to use built-in access methods as well as those created by the CREATE ACCESS METHOD statement of SQL that is described in the *IBM Informix Guide to SQL: Syntax*. The *sysams* table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>am_name</td>
<td>VARCHAR(128)</td>
<td>Name of the access method</td>
</tr>
<tr>
<td>am_owner</td>
<td>CHAR(32)</td>
<td>Name of the owner of the access method</td>
</tr>
<tr>
<td>am_id</td>
<td>INTEGER</td>
<td>Unique identifying code for an access method</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This corresponds to the <em>am_id</em> columns in the <em>systables</em>, <em>sysindices</em>, and <em>sysopclasses</em> tables.</td>
</tr>
<tr>
<td>am_type</td>
<td>CHAR(1)</td>
<td>Type of access method: P = Primary, S = Secondary</td>
</tr>
<tr>
<td>am_sptype</td>
<td>CHAR(3)</td>
<td>Types of spaces where the access method can exist: A or a = all types: extspaces, dbspaces, and sbspaces. If the access method is not user-defined (that is, if it is built in or registered during database creation by the server), it supports dbspaces. D or d = dbspaces only S or s = sbspaces only (smart-large-object space) X or x = extspaces only</td>
</tr>
<tr>
<td>am_defopclass</td>
<td>INTEGER</td>
<td>Unique identifying code for default-operator class</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value is the <em>opclassid</em> from the entry for this operator class in the <em>sysopclasses</em> table.</td>
</tr>
<tr>
<td>am_keyscan</td>
<td>INTEGER</td>
<td>Whether a secondary access method supports a key scan</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(An access method supports a key scan if it can return a key as well as a rowid from a call to the <em>am_getnext</em> function.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( 0 = FALSE; Non-zero = TRUE )</td>
</tr>
<tr>
<td>am_unique</td>
<td>INTEGER</td>
<td>Whether a secondary access method can support unique keys ( 0 = FALSE; Non-zero = TRUE )</td>
</tr>
<tr>
<td>am_cluster</td>
<td>INTEGER</td>
<td>Whether a primary access method supports clustering ( 0 = FALSE; Non-zero = TRUE )</td>
</tr>
<tr>
<td>am_rowids</td>
<td>INTEGER</td>
<td>Whether a primary access method supports rowids ( 0 = FALSE; Non-zero = TRUE )</td>
</tr>
<tr>
<td>am_readwrite</td>
<td>INTEGER</td>
<td>Whether a primary access method can both read and write ( 0 = access method is read-only Non-zero = access method is read/write)</td>
</tr>
<tr>
<td>am_parallel</td>
<td>INTEGER</td>
<td>Whether an access method supports parallel execution ( 0 = FALSE; Non-zero = TRUE )</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>am_costfactor</td>
<td>SMALLFLOAT</td>
<td>The value to be multiplied by the cost of a scan in order to normalize it to costing done for built-in access methods. The scan cost is the output of the am_scancost function.</td>
</tr>
<tr>
<td>am_create</td>
<td>INTEGER</td>
<td>The routine specified for the AM_CREATE purpose for this access method. Value = procid for the routine in the sysprocedures table.</td>
</tr>
<tr>
<td>am_drop</td>
<td>INTEGER</td>
<td>The routine specified for the AM_DROP purpose function for this access method.</td>
</tr>
<tr>
<td>am_open</td>
<td>INTEGER</td>
<td>The routine specified for the AM_OPEN purpose function for this access method.</td>
</tr>
<tr>
<td>am_close</td>
<td>INTEGER</td>
<td>The routine specified for the AM_CLOSE purpose function for this access method.</td>
</tr>
<tr>
<td>am_insert</td>
<td>INTEGER</td>
<td>The routine specified for the AM_INSERT purpose function for this access method.</td>
</tr>
<tr>
<td>am_delete</td>
<td>INTEGER</td>
<td>The routine specified for the AM_DELETE purpose function for this access method.</td>
</tr>
<tr>
<td>am_update</td>
<td>INTEGER</td>
<td>The routine specified for the AM_UPDATE purpose function for this access method.</td>
</tr>
<tr>
<td>am_stats</td>
<td>INTEGER</td>
<td>The routine specified for the AM_STATS purpose function for this access method.</td>
</tr>
<tr>
<td>am_scancost</td>
<td>INTEGER</td>
<td>The routine specified for the AM_SCANCOST purpose function for this access method.</td>
</tr>
<tr>
<td>am_check</td>
<td>INTEGER</td>
<td>The routine specified for the AM_CHECK purpose function for this access method.</td>
</tr>
<tr>
<td>am_beginscan</td>
<td>INTEGER</td>
<td>Routine specified for the AM_BEGINSCAN purpose function for this access method.</td>
</tr>
<tr>
<td>am_endscan</td>
<td>INTEGER</td>
<td>The routine specified for the AM_ENDSCAN purpose function for this access method.</td>
</tr>
<tr>
<td>am_rescan</td>
<td>INTEGER</td>
<td>The routine specified for the AM_RESCAN purpose function for this access method.</td>
</tr>
<tr>
<td>am_getnext</td>
<td>INTEGER</td>
<td>The routine specified for the AM_GETNEXT purpose function for this access method.</td>
</tr>
<tr>
<td>am_getbyid</td>
<td>INTEGER</td>
<td>The routine specified for the AM_GETBYID purpose function for this access method.</td>
</tr>
<tr>
<td>am_build</td>
<td>INTEGER</td>
<td>The routine specified for the AM_BUILD purpose function for this access method.</td>
</tr>
<tr>
<td>am_init</td>
<td>INTEGER</td>
<td>The routine specified for the AM_INIT purpose function for this access method.</td>
</tr>
<tr>
<td>am_truncate</td>
<td>INTEGER</td>
<td>The routine specified for the AM_TRUNCATE purpose function for this access method.</td>
</tr>
</tbody>
</table>

For each of the nearly 20 columns that follow am_costfactor, the value is the sysprocedures.procid value for the corresponding routine.

The am_sptype column can have multiple entries. For example:
• A means the access method supports extspaces and sbspaces. If the access method is built-in, such as a B-tree, it also supports dbspaces.
• DS means the access method supports dbspaces and sbspaces.
• sx means the access method supports sbspaces and extspaces.

A composite index on the am_name and am_owner columns in this table allows only unique values. The am_id column has a unique index.

For information about access method functions, refer to the documentation of your access method.

## SYSATTRTYPES (IDS)

The sysattrtypes system catalog table contains information about members of a complex data type. Each row of sysattrtypes contains information about elements of a collection data type or fields of a row data type.

The sysattrtypes table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>extended_id</td>
<td>INTEGER</td>
<td>Identifying code of an extended data type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Value is the same as in the sysxtdtypes table (page 1-56).</td>
</tr>
<tr>
<td>seqno</td>
<td>SMALLINT</td>
<td>Identifying code of an entry having extended_id type</td>
</tr>
<tr>
<td>levelno</td>
<td>SMALLINT</td>
<td>Position of member in collection hierarchy</td>
</tr>
<tr>
<td>parent_no</td>
<td>SMALLINT</td>
<td>Value in the seqno column of the complex data type that contains this member</td>
</tr>
<tr>
<td>fieldname</td>
<td>VARCHAR(128)</td>
<td>Name of the field in a row type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Null for other complex data types</td>
</tr>
<tr>
<td>fieldno</td>
<td>SMALLINT</td>
<td>Field number sequentially assigned by system (from left to right within each row type)</td>
</tr>
<tr>
<td>type</td>
<td>SMALLINT</td>
<td>Code for the data type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See the description of syscolumns.coltype (page 1-18).</td>
</tr>
<tr>
<td>length</td>
<td>SMALLINT</td>
<td>Length (in bytes) of the member</td>
</tr>
<tr>
<td>xtd_type_id</td>
<td>INTEGER</td>
<td>Code identifying this data type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>See the description of sysxtdtypes.extended_id (page 1-56).</td>
</tr>
</tbody>
</table>

Two indexes on the extended_id column and the xtd_type_id column allow duplicate values. A composite index on the extended_id and seqno columns allows only unique values.

## SYSBLOBS

The sysblobs system catalog table specifies the storage location of BYTE and TEXT column values. Its name is based on a legacy term for BYTE and TEXT columns, blobs (also known as simple large objects), and does not refer to the BLOB data type of Dynamic Server. The sysblobs table contains one row for each BYTE or TEXT column, and has the following columns.
### Column Types

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>spacename</td>
<td>VARCHAR(128)</td>
<td>Name of partition, dbspaces, or family</td>
</tr>
<tr>
<td>type</td>
<td>CHAR(1)</td>
<td>Code identifying the type of storage media:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>M = Magnetic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O = Optical (IDS)</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Code identifying the table</td>
</tr>
<tr>
<td>colno</td>
<td>SMALLINT</td>
<td>Column number within its table</td>
</tr>
</tbody>
</table>

A composite index on `tabid` and `colno` allows only unique values.

For information about the location and size of chunks of blobspaces, dbspaces, and sbspaces for TEXT, BYTE, BLOB, and CLOB columns, see the *IBM Informix Administrator's Guide* and the *IBM Informix Administrator's Reference*.

### SYSCASTS (IDS)

The `syscasts` system catalog table describes the casts in the database. It contains one row for each built-in cast, each implicit cast, and each explicit cast that a user defines. The `syscasts` table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>CHAR(32)</td>
<td>Owner of cast (user informix for built-in casts and user name for implicit and explicit casts)</td>
</tr>
<tr>
<td>argument_type</td>
<td>SMALLINT</td>
<td>Source data type on which the cast operates</td>
</tr>
<tr>
<td>argument_xid</td>
<td>INTEGER</td>
<td>Code for the source data type specified in the <code>argument_type</code> column</td>
</tr>
<tr>
<td>result_type</td>
<td>SMALLINT</td>
<td>Code for the data type returned by the cast</td>
</tr>
<tr>
<td>result_xid</td>
<td>INTEGER</td>
<td>Data type code of the data type named in the <code>result_type</code> column</td>
</tr>
<tr>
<td>routine_name</td>
<td>VARCHAR(128)</td>
<td>Function or procedure implementing the cast</td>
</tr>
<tr>
<td>routine_owner</td>
<td>CHAR(32)</td>
<td>Name of owner of the function or procedure specified in the <code>routine_name</code> column</td>
</tr>
<tr>
<td>class</td>
<td>CHAR(1)</td>
<td>Type of cast:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E = Explicit cast</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I = Implicit cast</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S = Built-in cast</td>
</tr>
</tbody>
</table>

If `routine_name` and `routine_owner` have NULL values, this indicates that the cast is defined without a routine. This can occur if both of the data types specified in the `argument_type` and `result_type` columns have the same length and alignment, and are passed by reference, or passed by value.

A composite index on columns `argument_type`, `argument_xid`, `result_type`, and `result_xid` allows only unique values. A composite index on columns `result_type` and `result_xid` allows duplicate values.
The `syschecks` system catalog table describes each check constraint defined in the database. Because the `syschecks` table stores both the ASCII text and a binary encoded form of the check constraint, it contains multiple rows for each check constraint. The `syschecks` table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>constrid</td>
<td>INTEGER</td>
<td>Unique code identifying the constraint</td>
</tr>
<tr>
<td>type</td>
<td>CHAR(1)</td>
<td>Form in which the check constraint is stored: B = Binary encoded, s = Select, T = Text</td>
</tr>
<tr>
<td>seqno</td>
<td>SMALLINT</td>
<td>Line number of the check constraint</td>
</tr>
<tr>
<td>checktext</td>
<td>CHAR(32)</td>
<td>Text of the check constraint</td>
</tr>
</tbody>
</table>

The text in the `checktext` column associated with B type in the `type` column is in computer-readable format. To view the text associated with a particular check constraint, use the following query with the appropriate `constrid` code:

```sql
SELECT * FROM syschecks WHERE constrid=10 AND type='T'
```

Each check constraint described in the `syschecks` table also has its own row in the `sysconstraints` table.

A composite index on the `constrid`, `type`, and `seqno` columns allows only unique values.

The `syscheckudrdep` system catalog table describes each check constraint that is referenced by a user-defined routine (UDR) in the database. The `syscheckudrdep` table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>udr_id</td>
<td>INTEGER</td>
<td>Unique code identifying the UDR</td>
</tr>
<tr>
<td>constraint_id</td>
<td>INTEGER</td>
<td>Unique code identifying the check constraint</td>
</tr>
</tbody>
</table>

Each check constraint described in the `syscheckudrdep` table also has its own row in the `sysconstraints` system catalog table, where the `constrid` column has the same value as the `constraint_id` column of `syscheckudrdep`.

A composite index on the `udr_id` and `constraint_id` columns requires that combinations of these values be unique.

The `syscolattrs` system catalog table describes the characteristics of smart large objects, namely CLOB and BLOB data types. It contains one row for each subspace listed in the PUT clause of the CREATE TABLE statement.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Code uniquely identifying the table</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Explanation</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>colno</td>
<td>SMALLINT</td>
<td>Number of the column that contains the smart large object</td>
</tr>
<tr>
<td>extensize</td>
<td>INTEGER</td>
<td>Pages in smart-large-object extent, expressed in kilobytes</td>
</tr>
<tr>
<td>flags</td>
<td>INTEGER</td>
<td>Integer representation of the combination (by addition) of hexadecimal values of the following parameters:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LO_NOLOG ( (0x00000001 = 1) )  The smart large object is not logged.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LO_LOG ( (0x00000010 = 2) )  Logging of smart large objects conforms to current log mode of the database.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LO_KEEP_LASTACCESS_TIME ( (0x00000100 = 4) )  A record is kept of the most recent access of this smart-large-object column by a user.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LO_NOKEEP_LASTACCESS_TIME ( (0x00001000 = 8) )  No record is kept of the most recent access of this smart-large-object column by a user.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HI_INTEG ( (0x00010000 = 16) )  Data pages have headers and footers to detect incomplete writes and data corruption.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MODERATE_INTEG ( \text{Not available at this time} )  Data pages do not have headers and footers.</td>
</tr>
<tr>
<td>flags1</td>
<td>INTEGER</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>sbspace</td>
<td>VARCHAR(128)</td>
<td>Name of the sbspace</td>
</tr>
</tbody>
</table>

A composite index on the **tabid**, **colno**, and **sbspace** columns allows only unique combinations of these values.

**SYSCOLAUTH**

The **syscolauth** system catalog table describes each set of privileges granted on a column. It contains one row for each set of column privileges granted in the database. The **syscolauth** table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantor</td>
<td>VARCHAR(32)</td>
<td>Name of the grantor of privilege</td>
</tr>
<tr>
<td>grantee</td>
<td>VARCHAR(32)</td>
<td>Name of the grantee of privilege</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Code uniquely identifying the table</td>
</tr>
<tr>
<td>colno</td>
<td>SMALLINT</td>
<td>Column number within its table</td>
</tr>
<tr>
<td>colauth</td>
<td>CHAR(3)</td>
<td>3-byte pattern that specifies column privileges:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>s or S = Select</td>
</tr>
<tr>
<td></td>
<td></td>
<td>u or U = Update</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r or R = References</td>
</tr>
</tbody>
</table>

If the **colauth** privilege code is uppercase (for example, S for Select), a user who has this privilege can also grant it to others. If the **colauth** privilege code is lowercase (for example, s for Select), the user who has this privilege cannot grant it to others. A hyphen ( - ) indicates the absence of the privilege corresponding to that position within the **colauth** pattern.
A composite index on the tabid, grantor, grantee, and colno columns allows only unique values. A composite index on the tabid and grantee columns allows duplicate values.

**SYSCOLDEPEND**

The syscoldepend system catalog table tracks the table columns specified in check and NOT NULL constraints. Because a check constraint can involve more than one column in a table, the syscoldepend table can contain multiple rows for each check constraint; one row is created for each column involved in the constraint. The syscoldepend table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>constrid</td>
<td>INTEGER</td>
<td>Code uniquely identifying the constraint</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Code uniquely identifying the table</td>
</tr>
<tr>
<td>colno</td>
<td>SMALLINT</td>
<td>Column number within the table</td>
</tr>
</tbody>
</table>

A composite index on the constrid, tabid, and colno columns allows only unique values. A composite index on the tabid and colno columns allows duplicate values.

See also the syscheckudrdep system catalog table in “SYSCHECKUDRDEP (IDS)” on page 1-15 which lists every check constraint that is referenced by a user-defined routine.

See also the sysnewdepend table in “SYSNEWDEPEND (XPS)” on page 1-33 which describes the column dependencies of generalized-key indexes.

See also the sysreferences table in “SYSREFERENCES” on page 1-41 which describes dependencies of referential constraints.
The `syscolumns` system catalog table describes each column in the database. One row exists for each column that is defined in a table or view.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>colname</td>
<td>VARCHAR(128)</td>
<td>Column name</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Identifying code of table containing the column</td>
</tr>
<tr>
<td>colno</td>
<td>SMALLINT</td>
<td>Column number</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The system sequentially assigns this (from left to right within each table).</td>
</tr>
<tr>
<td>coltype</td>
<td>SMALLINT</td>
<td>Code indicating the data type of the column:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = CHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = SMALLINT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 = INTEGER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3 = FLOAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 = SMALLFLOAT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5 = DECIMAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6 = SERIAL *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7 = DATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 = MONEY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9 = NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 = DATETIME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11 = BYTE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12 = TEXT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>13 = VARCHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>14 = INTERVAL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>15 = NCHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 = NVARCHAR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17 = INT8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18 = SERIAL8 *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19 = SET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20 = MULTISET</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21 = LIST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22 = Unnamed ROW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>40 = Variable-length</td>
</tr>
<tr>
<td></td>
<td></td>
<td>41 = Named ROW</td>
</tr>
<tr>
<td>collength</td>
<td>SMALLINT</td>
<td>Column length (in bytes)</td>
</tr>
<tr>
<td>colmin</td>
<td>INTEGER</td>
<td>Minimum column length (in bytes)</td>
</tr>
<tr>
<td>colmax</td>
<td>INTEGER</td>
<td>Maximum column length (in bytes)</td>
</tr>
<tr>
<td>extended_id (IDS)</td>
<td>INTEGER</td>
<td>Data type code, from the <code>sysxtdtypes</code> table, of the data type specified in the <code>coltype</code> column</td>
</tr>
<tr>
<td>SECLABELID</td>
<td>INTEGER</td>
<td>The label ID of the security label associated with the column if it is a protected column. Null otherwise.</td>
</tr>
</tbody>
</table>

* In DB–Access, an offset value of 256 is always added to these `coltype` codes because DB–Access sets SERIAL and SERIAL8 columns to NOT NULL.

Extended Parallel Server does not support opaque data types, nor the complex data types SET, MULTISET, LIST, unnamed and named ROW.

A composite index on `tabid` and `colno` allows only unique values.

The `coltype` codes listed on the previous page can be incremented by bitmaps showing the following features of the column.

**Bit Value** | **Significance When Bit Is Set** |
---|---|
0x0100 | NULL values are not allowed |
0x0200 | Value is from a host variable |
0x0400 | Float-to-decimal for networked database server |
0x0800 | DISTINCT data type |
0x1000 | Named ROW type |
0x2000 | DISTINCT type from LVARCHAR base type |
0x4000  DISTINCT type from BOOLEAN base type
0x8000  Collection is processed on client system

For example, the coltype value 4118 for named row types is the decimal representation of the hexadecimal value 0x1016, which is the same as the hexadecimal coltype value for an unnamed row type (0x016), with the named-row-type bit set. (The file $INFORMIXDIR/incl/esql/sqltypes.h contains additional information about syscolumns.coltype codes.)

**NOT NULL Constraints**
Similarly, the coltype value is incremented by 256 if the column does not allow NULL values. To determine the data type for such columns, subtract 256 from the value and evaluate the remainder, based on the possible coltype values. For example, if the coltype value is 262, subtracting 256 leaves a remainder of 6, indicating that the column has a SERIAL data type.

**Storing the Column Data Type**
The database server stores the coltype value as bitmap, as listed in “SYSCOLUMNS” on page 1-18.

**Opaque Data Types (IDS)**
The BOOLEAN, BLOB, CLOB, and LVARCHAR data types are implemented by the database server as built-in opaque data types.

A built-in opaque data type is one for which the database server provides the type definition. Because these data types are built-in opaque types, they do not have a unique coltype value. Instead, they have one of the coltype values for opaque types: 41 (fixed-length opaque type), or 40 (varying-length opaque type). The different fixed-length opaque types are distinguished by the extended_id column in the sysxtdtypes system catalog table.

The following are the coltype values for the built-in opaque data types.

<table>
<thead>
<tr>
<th>Predefined Data Type</th>
<th>Value for coltype Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB</td>
<td>41</td>
</tr>
<tr>
<td>CLOB</td>
<td>41</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>41</td>
</tr>
<tr>
<td>LVARCHAR</td>
<td>40</td>
</tr>
</tbody>
</table>

**Storing Column Length**
The collength column value depends on the data type of the column.

**Integer-Based Data Types**
A collength value for a BIGINT, DATE, INTEGER, INT8, SERIAL, SERIAL8, or SMALLINT column is machine-independent. The database server uses the following lengths for these integer-based data types of the SQL language.
### Integer-Based Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Length (in Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALLINT</td>
<td>2</td>
</tr>
<tr>
<td>DATE</td>
<td>4</td>
</tr>
<tr>
<td>INTEGER</td>
<td>4</td>
</tr>
<tr>
<td>SERIAL</td>
<td>4</td>
</tr>
<tr>
<td>BIGINT, INT8</td>
<td>8 (XPS)</td>
</tr>
<tr>
<td>SERIAL8</td>
<td>8 (XPS)</td>
</tr>
<tr>
<td>INT8</td>
<td>10 (IDS)</td>
</tr>
<tr>
<td>SERIAL8</td>
<td>10 (IDS)</td>
</tr>
</tbody>
</table>

### Varying-Length Character Data Types

For Dynamic Server columns of the LVARCHAR type, `collength` has the value of `max` from the data type declaration, or 2048 if no maximum was specified.

For VARCHAR or NVARCHAR columns, the `max_size` and `min_space` values are encoded in the `collength` column using one of these formulas:

- If the `collength` value is positive:
  \[
  collength = (min\_space \times 256) + max\_size
  \]
- If the `collength` value is negative:
  \[
  collength + 65536 = (min\_space \times 256) + max\_size
  \]

### Time Data Types

As noted previously, DATE columns have a value of 4 in the `collength` column.

For columns of type DATETIME or INTERVAL, `collength` is determined using the following formula:

\[
(length \times 256) + (first\_qualifier \times 16) + last\_qualifier
\]

The length is the physical length of the DATETIME or INTERVAL field, and `first\_qualifier` and `last\_qualifier` have values that the following table shows.

<table>
<thead>
<tr>
<th>Field Qualifier</th>
<th>Value</th>
<th>Field Qualifier</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR</td>
<td>0</td>
<td>FRACTION(1)</td>
<td>11</td>
</tr>
<tr>
<td>MONTH</td>
<td>2</td>
<td>FRACTION(2)</td>
<td>12</td>
</tr>
<tr>
<td>DAY</td>
<td>4</td>
<td>FRACTION(3)</td>
<td>13</td>
</tr>
<tr>
<td>HOUR</td>
<td>6</td>
<td>FRACTION(4)</td>
<td>14</td>
</tr>
<tr>
<td>MINUTE</td>
<td>8</td>
<td>FRACTION(5)</td>
<td>15</td>
</tr>
<tr>
<td>SECOND</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For example, if a DATETIME YEAR TO MINUTE column has a length of 12 (such as YYYY:DD:MO:HH:MI), a `first\_qualifier` value of 0 (for YEAR), and a `last\_qualifier` value of 8 (for MINUTE), then the `collength` value is 3080 (from \((256 \times 12) + (0 \times 16) + 8\)).

### Fixed-Point Data Types

The `collength` value for a MONEY or DECIMAL \((p, s)\) column can be calculated using the following formula:

\[
(precision \times 256) + scale
\]
Simple-Large-Object Data Types
If the data type of the column is BYTE or TEXT, collength holds the length of the descriptor.

Storing Maximum and Minimum Values
The colmin and colmax column values hold the second-smallest and second-largest data values in the column, respectively. For example, if the values in an indexed column are 1, 2, 3, 4, and 5, the colmin value is 2 and the colmax value is 4. Storing the second-smallest and second-largest data values lets the database server make assumptions about the range of values in a given column and, in turn, further optimize search strategies.

The colmin and colmax columns contain values only if the column is indexed and you have run the UPDATE STATISTICS statement. If you store BYTE or TEXT data in the tblspace, the colmin value is -1.

The colmin and colmax columns are valid only for data types that fit into four bytes: SMALLFLOAT, SMALLINT, INTEGER, and the first four bytes of CHAR. The values for all other noninteger column types are the initial four bytes of the maximum or minimum value, which are treated as integers.

It is better to use UPDATE STATISTICS MEDIUM than to depend on colmin and colmax values. UPDATE STATISTICS MEDIUM gives better information and is valid for all data types.

Dynamic Server does not calculate colmin and colmax values for user-defined data types. These columns, however, have values for user-defined data types if a user-defined secondary access method supplies them.

**SYSCONSTRAINTS**
The sysconstraints system catalog table lists the constraints placed on the columns in each database table. An entry is also placed in the sysindexes system catalog table (or sysindices view for Dynamic Server) for each unique, primary key, or referential constraint that does not already have a corresponding entry in sysindexes or sysindices. Because indexes can be shared, more than one constraint can be associated with an index. The sysconstraints table has the following columns.
### SYSDEFAULTS

The sysdefaults system catalog table lists the user-defined defaults that are placed on each column in the database. One row exists for each user-defined default value. The sysdefaults table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Code uniquely identifying a table</td>
</tr>
<tr>
<td>colno</td>
<td>SMALLINT</td>
<td>Code uniquely identifying a column</td>
</tr>
<tr>
<td>type</td>
<td>CHAR(1)</td>
<td>Code identifying the type of default value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C = Current</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L = Literal value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N = NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S = DBservername or Sitename</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T = Today</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U = Today</td>
</tr>
<tr>
<td>default</td>
<td>CHAR(256)</td>
<td>If sysdefaults.type = L, a literal default value</td>
</tr>
<tr>
<td>class (IDS)</td>
<td>CHAR(1)</td>
<td>Code identifying what kind of column:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T = table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t = ROW type</td>
</tr>
</tbody>
</table>

If no default is specified explicitly in the CREATE TABLE or the ALTER TABLE statement, then no entry exists for that column in the sysdefaults table.

If you specify a literal for the default value, it is stored in the default column as ASCII text. If the literal value is not of one of the data types listed in the next paragraph, the default column consists of two parts. The first part is the 6-bit
representation of the binary value of the default value structure. The second part is the default value in ASCII text. A blank space separates the two parts.

If the data type of the column is not CHAR, NCHAR, NVARCHAR, or VARCHAR, or (for Dynamic Server) BOOLEAN or LVARCHAR, a binary representation of the default value is encoded in the default column.

A composite index on the tabid, colno, and class columns allows only unique values. (For Extended Parallel Server, this index omits the class column.)

---

SYSDEPEND

The **sysdepend** system catalog table describes how each view or table depends on other views or tables. One row exists in this table for each dependency, so a view based on three tables has three rows. The **sysdepend** table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>btabid</td>
<td>INTEGER</td>
<td>Code uniquely identifying the base table or view</td>
</tr>
<tr>
<td>btype</td>
<td>CHAR(1)</td>
<td>Base object type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T = Table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>V = View</td>
</tr>
<tr>
<td>dtabid</td>
<td>INTEGER</td>
<td>Code uniquely identifying a dependent table or view</td>
</tr>
<tr>
<td>dtype</td>
<td>CHAR(1)</td>
<td>Code for the type of dependent object; currently, only view (V = View) is implemented</td>
</tr>
</tbody>
</table>

The btabid and dtabid columns are indexed and allow duplicate values.

---

SYSDIRECTIVES (IDS)

The **sysdirectives** table stores external optimizer directives that can be applied to queries. Whether queries in client applications can use these optimizer directives depends on the setting of the **IFX_EXTDIRECTIVES** environment variable on the client system, as described in Chapter 3, and on the EXT_DIRECTIVES setting in the configuration file of the database server.

The **sysdirectives** table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>SERIAL</td>
<td>Unique code identifying the optimizer directive</td>
</tr>
<tr>
<td>query</td>
<td>TEXT</td>
<td>Text of the query as it appears in the application</td>
</tr>
<tr>
<td>directives</td>
<td>TEXT</td>
<td>Text of the optimizer directive, without comments</td>
</tr>
<tr>
<td>active</td>
<td>SMALLINT</td>
<td>Integer code that identifies whether this entry is active (= 1) or test only (= 2)</td>
</tr>
<tr>
<td>hash_code</td>
<td>SMALLINT</td>
<td>For internal use only</td>
</tr>
</tbody>
</table>

NULL values are not valid in the query column. There is a unique index on the id column.
SYSDISTRIBUT

The sysdistrib system catalog table stores data-distribution information for the database server to use. Data distributions provide detailed table and column information to the optimizer to improve the choice of execution paths of SELECT statements. The sysdistrib table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Code uniquely identifying the table where data values were gathered</td>
</tr>
<tr>
<td>colno</td>
<td>SMALLINT</td>
<td>Column number in the source table</td>
</tr>
<tr>
<td>seqno</td>
<td>INTEGER</td>
<td>Ordinal number for multiple entries</td>
</tr>
<tr>
<td>constructed</td>
<td>YEAR TO FRACTION(5)</td>
<td>Date when the data distribution was created</td>
</tr>
<tr>
<td>mode</td>
<td>CHAR(1)</td>
<td>Optimization level: M = Medium H = High</td>
</tr>
<tr>
<td>resolution</td>
<td>SMALLFLOAT</td>
<td>Specified in the UPDATE STATISTICS statement</td>
</tr>
<tr>
<td>confidence</td>
<td>SMALLFLOAT</td>
<td>Specified in the UPDATE STATISTICS statement</td>
</tr>
<tr>
<td>encdat</td>
<td>STAT(256) CHAR(256) XPS</td>
<td>Statistics information</td>
</tr>
<tr>
<td>type (IDS)</td>
<td>CHAR(1)</td>
<td>Type of statistics: A = encdat has ASCII-encoded histogram in fixed-length character field</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S = encdat has user-defined statistics</td>
</tr>
<tr>
<td>smplsize (IDS)</td>
<td>SMALLFLOAT</td>
<td>A value greater than zero up to 1.0 indicating a percentage of the total rows in the table that USTATS samples. Values greater than 1.0 indicate the actual number of rows used that USTATS samples. A value of zero indicates that the sample size is not specified. USTATS HIGH always updates all rows.</td>
</tr>
</tbody>
</table>

Information is stored in the sysdistrib table when an UPDATE STATISTICS statement with mode MEDIUM or HIGH is executed for a table. (UPDATE STATISTICS LOW does not insert a value into the mode column.)

Only user informix can select the encdat column.

Each row in the sysdistrib system catalog table is keyed by the tabid and colno for which the statistics are collected.

For built-in data type columns, the type field is set to A. The encdat column stores an ASCII-encoded histogram that is broken down into multiple rows, each of which contains 256 bytes.

In Dynamic Server, for columns of user-defined data types, the type field is set to S. The encdat column stores the statistics collected by the statcollect user-defined routine in multirepresentational form. Only one row is stored for each tabid and colno pair. A composite index on the tabid, colno, and seqno columns requires unique combinations of values.
The sysdomains view is not used. It displays columns of other system catalog tables. It has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>SERIAL</td>
<td>Unique code identifying the domain</td>
</tr>
<tr>
<td>owner</td>
<td>CHAR(32)</td>
<td>Name of the owner of the domain</td>
</tr>
<tr>
<td>name</td>
<td>VARCHAR(128)</td>
<td>Name of the domain</td>
</tr>
<tr>
<td>type</td>
<td>SMALLINT</td>
<td>Code identifying the type of domain</td>
</tr>
</tbody>
</table>

There is no index on this view.

The syserrors system catalog table stores information about error, warning, and informational messages returned by DataBlade™ modules and user-defined routines using the mi_db_error_raise() DataBlade API function.

The syserrors table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>sqlstate</td>
<td>CHAR(5)</td>
<td>SQLSTATE value associated with the error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information about SQLSTATE values and their meanings, see the GET DIAGNOSTICS statement in the IBM Informix Guide to SQL: Syntax.</td>
</tr>
<tr>
<td>locale</td>
<td>CHAR(36)</td>
<td>The locale with which this version of the message is associated (for example, en_us.8859-1)</td>
</tr>
<tr>
<td>level</td>
<td>SMALLINT</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>seqno</td>
<td>SMALLINT</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>message</td>
<td>VARCHAR(255)</td>
<td>Message text</td>
</tr>
</tbody>
</table>

To create a new message, insert a row directly into the syserrors table. By default, all users can view this table, but only users with the DBA privilege can modify it.

A composite index on the sqlstate, locale, level, and seqno columns allows only unique values.
The `sysexcols` system catalog table contains a row that describes each of the internal columns in external table `tabid` of format type (`fmttype`) FIXED. The `sysexcols` table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tabid</code></td>
<td>INTEGER</td>
<td>Unique identifying code of a table</td>
</tr>
<tr>
<td><code>colno</code></td>
<td>SMALLINT</td>
<td>Code identifying the column</td>
</tr>
<tr>
<td><code>exttype</code></td>
<td>SMALLINT</td>
<td>Code identifying an external column type</td>
</tr>
<tr>
<td><code>extstart</code></td>
<td>SMALLINT</td>
<td>Starting position of column in the external data file</td>
</tr>
<tr>
<td><code>extlength</code></td>
<td>SMALLINT</td>
<td>External column length (in bytes)</td>
</tr>
<tr>
<td><code>nullstr</code></td>
<td>CHAR(256)</td>
<td>Represents NULL in external data</td>
</tr>
<tr>
<td><code>picture</code></td>
<td>CHAR(256)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td><code>decimal</code></td>
<td>SMALLINT</td>
<td>Precision for external decimals</td>
</tr>
<tr>
<td><code>exttype</code></td>
<td>VARCHAR(128)</td>
<td>External type name</td>
</tr>
</tbody>
</table>

No entries are stored in `sysexcols` for DELIMITED or Informix format external files.

You can use the DBSCHEMA utility to write out the description of the external tables. To query these system catalog tables about an external table, use the `tabid` as stored in `systables` with `tabtype` = ‘E’.

An index on the `tabid` column allows duplicate values.

For each external table, at least one row exists in the `sysextdfiles` system catalog table, which has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tabid</code></td>
<td>INTEGER</td>
<td>Unique identifying code of an external table</td>
</tr>
<tr>
<td><code>dfentry</code></td>
<td>CHAR(152)</td>
<td>Data file entry</td>
</tr>
</tbody>
</table>

You can use DBSCHEMA to write out the description of the external tables. To query these system catalog tables about an external table, use the `tabid` as stored in `systables` with `tabtype` = ‘E’.

An index on the `tabid` column allows duplicate values.
SYSEXTernal (XPS)

For each external table, a single row exists in the sysexternal system catalog table. The `tabid` column associates the external table in this system catalog table with an entry in `systables`.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Unique identifying code of an external table</td>
</tr>
<tr>
<td>fmttype</td>
<td>CHAR(1)</td>
<td>Type of format: D = (delimited) F = (fixed) I = (Informix)</td>
</tr>
<tr>
<td>codeset</td>
<td>VARCHAR(128)</td>
<td>ASCII, EBCDIC</td>
</tr>
<tr>
<td>recdelim</td>
<td>CHAR(4)</td>
<td>The record delimiter</td>
</tr>
<tr>
<td>fddelim</td>
<td>CHAR(4)</td>
<td>The field delimiter</td>
</tr>
<tr>
<td>datefmt</td>
<td>CHAR(8)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>moneyfmt</td>
<td>CHAR(20)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>maxerrors</td>
<td>INTEGER</td>
<td>Number of errors to allow per coserver</td>
</tr>
<tr>
<td>rejectfile</td>
<td>CHAR(128)</td>
<td>Name of reject file</td>
</tr>
<tr>
<td>flags</td>
<td>INTEGER</td>
<td>Optional load flags</td>
</tr>
<tr>
<td>ndfiles</td>
<td>INTEGER</td>
<td>Number of data files in sysextfiles</td>
</tr>
</tbody>
</table>

You can use DBSCHEMA to write out the description of the external tables. To query these system catalog tables about an external table, use the `tabid` as stored in `systables` with `tabtype` = 'E'.

An index on the `tabid` column allows only unique values.

SYSFRAGAUTH (IDS)

The `sysfragauth` system catalog table stores information about the privileges that are granted on table fragments. This table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantor</td>
<td>CHAR(32)</td>
<td>Name of the grantor of privilege</td>
</tr>
<tr>
<td>grantee</td>
<td>CHAR(32)</td>
<td>Name of the grantee of privilege</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Identifying code of the fragmented table</td>
</tr>
<tr>
<td>fragment</td>
<td>VARCHAR(128)</td>
<td>Name of dbspace where fragment is stored</td>
</tr>
<tr>
<td>fragauth</td>
<td>CHAR(6)</td>
<td>A 6-byte pattern specifying fragment privileges (including 3 bytes reserved for future use): u or U = Update i or I = Insert d or D = Delete</td>
</tr>
</tbody>
</table>

In the `fragauth` column, an uppercase code (such as U for Update) means that the grantee can grant the privilege to other users; a lowercase (for example, u for Update) means the user cannot grant the privilege to others. Hyphen ( - ) indicates the absence of the privilege for that position within the pattern.
A composite index on the **tabid**, **grantor**, **grantee**, and **fragment** columns allows only unique values. A composite index on the **tabid** and **grantee** columns allows duplicate values.

The following example displays the fragment-level privileges for one base table, as they appear in the **sysfragauth** table. In this example, the grantee **ted** can grant the Update, Delete, and Insert privileges to other users.

<table>
<thead>
<tr>
<th>grantor</th>
<th>grantee</th>
<th>tabid</th>
<th>fragment</th>
<th>fragauth</th>
</tr>
</thead>
<tbody>
<tr>
<td>dba</td>
<td>dick</td>
<td>101</td>
<td>dbsp1</td>
<td>-ui---</td>
</tr>
<tr>
<td>dba</td>
<td>jane</td>
<td>101</td>
<td>dbsp3</td>
<td>--i----</td>
</tr>
<tr>
<td>dba</td>
<td>mary</td>
<td>101</td>
<td>dbsp4</td>
<td>--id--</td>
</tr>
<tr>
<td>dba</td>
<td>ted</td>
<td>101</td>
<td>dbsp2</td>
<td>-UID--</td>
</tr>
</tbody>
</table>

**SYSFRAGMENTS**

The **sysfragments** system catalog table stores fragmentation information for tables and indexes. One row exists for each table or index fragment.

The **sysfragments** table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>fragtype</td>
<td>CHAR(1)</td>
<td>Code indicating the type of fragmented object:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I = Original index fragment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i = Duplicated index fragment (XPS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T = Original table fragment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t = Duplicated table fragment (XPS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B = TEXT or BYTE data (XPS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i = Index fragments of a duplicated table (XPS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d = Data fragments of a duplicated table (XPS)</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Unique identifying code of table</td>
</tr>
<tr>
<td>indexname</td>
<td>VARCHAR(128)</td>
<td>Name of index</td>
</tr>
<tr>
<td>colno</td>
<td>INTEGER</td>
<td>Identifying code of TEXT or BYTE column (IDS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Identifying code of replica identifier (XPS)</td>
</tr>
<tr>
<td>partn</td>
<td>INTEGER</td>
<td>Identifying code of physical location</td>
</tr>
<tr>
<td>strategy</td>
<td>CHAR(1)</td>
<td>Code for type of fragment distribution strategy:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R = Round-robin fragmentation strategy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E = Expression-based fragmentation strategy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I = IN DBSPACE clause specifies a specific location as part of fragmentation strategy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T = Table-based fragmentation strategy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>H = Table is a subtable within a table hierarchy (IDS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hash-based fragmentation strategy (XPS)</td>
</tr>
<tr>
<td>location</td>
<td>CHAR(1)</td>
<td>Reserved for future use; shows L for local</td>
</tr>
<tr>
<td>servername</td>
<td>VARCHAR(128)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>evalpos</td>
<td>INTEGER</td>
<td>Position of fragment in the fragmentation list</td>
</tr>
<tr>
<td>exprtext</td>
<td>TEXT</td>
<td>Expression for fragmentation strategy (IDS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Contains names of the columns that are hashed and composite information for hybrid fragmentation strategies; shows hashed columns followed by the fragmentation expression of the dbslice. (XPS)</td>
</tr>
</tbody>
</table>
In certain situations, you can duplicate selected tables across coservers to improve performance. If you have a duplicate copy of a small table on each coserver, then the database server can execute some small queries (queries that do not need rows from a table fragment on any other coserver and that do not require more than 128 kilobytes of memory per operator) as serial plans instead of as parallel plans that the Resource Grant Manager (RGM) manages. This performance feature applies only to OLTP-type transactions.

The following query returns the owner and name for each of the duplicated tables in the current database:

```sql
SELECT DISTINCT st.owner, st.tabname
FROM systables st, sysfragments sf
WHERE st.tabid = sf.tabid AND sf.fragtype = 't'
```

For more information about duplicating tables, refer to the description of the CREATE DUPLICATE statement in *IBM Informix Guide to SQL: Syntax*.

The *strategy* type T is used for attached indexes. (This is a fragmented index whose fragmentation is the same as the table fragmentation.)

In Dynamic Server, a composite index on the *fragtype*, *tabid*, *indexname*, and *evalpos* columns allows duplicate values.
In Extended Parallel Server, a composite index on the fragtype, tabid, indexname, evalpos, and hybdpos columns allows duplicate values.

SYSINDEXES

The sysindexes table is a view on the sysindices table. It contains one row for each index in the database. The sysindexes table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>idxname</td>
<td>VARCHAR(128)</td>
<td>Index name</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR(32)</td>
<td>Owner of index (user informix for system catalog tables and username for database tables) (IDS)</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Unique identifying code of table</td>
</tr>
<tr>
<td>idxtype</td>
<td>CHAR(1)</td>
<td>Index type: U= Unique D=Duplicates allowed G=Nonbitmap generalized-key index (XPS) g=Bitmap generalized-key index u=unique, bitmap d=nonunique, bitmap</td>
</tr>
<tr>
<td>clustered</td>
<td>CHAR(1)</td>
<td>Clustered or nonclustered index (C=Clustered)</td>
</tr>
<tr>
<td>part1</td>
<td>SMALLINT</td>
<td>Column number (colno) of a single index or the 1st component of a composite index</td>
</tr>
<tr>
<td>part2</td>
<td>SMALLINT</td>
<td>2nd component of a composite index</td>
</tr>
<tr>
<td>part3</td>
<td>SMALLINT</td>
<td>3rd component of a composite index</td>
</tr>
<tr>
<td>part4</td>
<td>SMALLINT</td>
<td>4th component of a composite index</td>
</tr>
<tr>
<td>part5</td>
<td>SMALLINT</td>
<td>5th component of a composite index</td>
</tr>
<tr>
<td>part6</td>
<td>SMALLINT</td>
<td>6th component of a composite index</td>
</tr>
<tr>
<td>part7</td>
<td>SMALLINT</td>
<td>7th component of a composite index</td>
</tr>
<tr>
<td>part8</td>
<td>SMALLINT</td>
<td>8th component of a composite index</td>
</tr>
<tr>
<td>part9</td>
<td>SMALLINT</td>
<td>9th component of a composite index</td>
</tr>
<tr>
<td>part10</td>
<td>SMALLINT</td>
<td>10th component of a composite index</td>
</tr>
<tr>
<td>part11</td>
<td>SMALLINT</td>
<td>11th component of a composite index</td>
</tr>
<tr>
<td>part12</td>
<td>SMALLINT</td>
<td>12th component of a composite index</td>
</tr>
<tr>
<td>part13</td>
<td>SMALLINT</td>
<td>13th component of a composite index</td>
</tr>
<tr>
<td>part14</td>
<td>SMALLINT</td>
<td>14th component of a composite index</td>
</tr>
<tr>
<td>part15</td>
<td>SMALLINT</td>
<td>15th component of a composite index</td>
</tr>
<tr>
<td>part16</td>
<td>SMALLINT</td>
<td>16th component of a composite index</td>
</tr>
<tr>
<td>levels</td>
<td>SMALLINT</td>
<td>Number of B-tree levels</td>
</tr>
<tr>
<td>leaves</td>
<td>INTEGER</td>
<td>Number of leaves</td>
</tr>
<tr>
<td>nunique</td>
<td>INTEGER</td>
<td>Number of unique keys in the first column</td>
</tr>
<tr>
<td>clust</td>
<td>INTEGER</td>
<td>Degree of clustering; smaller numbers correspond to greater clustering</td>
</tr>
<tr>
<td>idxflags</td>
<td>INTEGER</td>
<td>Bitmap storing the current locking mode of the index Normal = 0x00000001 (XPS) Coarse = 0x00000002 (XPS)</td>
</tr>
</tbody>
</table>
As with most system catalog tables, changes that affect existing indexes are reflected in this table only after you run the UPDATE STATISTICS statement.

Each part1 through part16 column in this table holds the column number (colno) of one of the 16 possible parts of a composite index. If the component is ordered in descending order, the colno is entered as a negative value. The columns are filled in for B-tree indexes that do not use user-defined data types or functional indexes. For generic B-trees and all other access methods, the part1 through part16 columns all contain zeros.

The clust column is blank until the UPDATE STATISTICS statement is run on the table. The maximum value is the number of rows in the table, and the minimum value is the number of data pages in the table.

In Extended Parallel Server, the tabid column is indexed and allows duplicate values. A composite index on the idxname, owner, and tabid columns allows only unique values.

### SYSINDICES (IDS)

The sysindices system catalog table describes the indexes in the database. It contains one row for each index that is defined in the database.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>idxname</td>
<td>VARCHAR(128)</td>
<td>Name of index</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR(32)</td>
<td>Name of owner of index (user informix for system catalog tables and username for database tables)</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Unique identifying code of table</td>
</tr>
</tbody>
</table>
| idxtype| CHAR(1)           | Index type:
|        |                   | U = Unique
|        |                   | D = Duplicates allowed                                                      |
| clustered| CHAR(1)      | Clustered or nonclustered index
|        |                   | (C = Clustered)                                                            |
| levels | SMALLINT          | Number of tree levels                                                       |
| leaves | INTEGER           | Number of leaves                                                            |
| numunique| INTEGER          | Number of unique keys in the first column                                  |
| clust  | INTEGER           | Degree of clustering; smaller numbers correspond to greater clustering. The maximum value is the number of rows in the table, and the minimum value is the number of data pages in the table. This column is blank until UPDATE STATISTICS is run on the table. |
| nrows  | FLOAT             | Estimated number of rows in the table (zero until UPDATE STATISTICS is run on the table). |
| indexkeys| INDEX-KEYARRAY | Column can have up to three fields, in the format: procid, (col1, ... , coln), opclassid where 1 < n < 341 |
| amid   | INTEGER           | Unique identifying code of the access method that implements this index. (Value = am_id for that access method in the sysams table.) |
| amparam| LVARCHAR          | List of parameters used to customize the behavior of this access method.    |
| collation| CHAR(32)       | Collating order at the time of index creation.                             |
Tip: This system catalog table is changed from Version 7.2 of Informix database servers. The earlier schema of this system catalog table is still available as a view and can be accessed under its original name: sysindexes.

Changes that affect existing indexes are reflected in this system catalog table only after you run the UPDATE STATISTICS statement.

The fields within the indexkeys columns have the following significance:
- The procid (as in sysprocedures) appears only for a functional index on return values of a function defined on columns of the table.
- The list of columns (col1, ..., coln) in the second field identifies the columns on which the index is defined. The maximum is language-dependent: up to 341 for an SPL or Java™ UDR; up to 102 for a C UDR.
- The opclassid identifies the secondary access method that the database server used to build and to search the index. This is the same as the sysopclasses.opclassid value for the access method.

The tabid column is indexed and allows duplicate values. A composite index on the idxname, owner, and tabid columns allows only unique values.

### SYSINHERITS (IDS)

The sysinherits system catalog table stores information about table and named ROW type inheritance. Every supertype, subtype, supertable, and subtable in the database has a corresponding row in the sysinherits table.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>child</td>
<td>INTEGER</td>
<td>Identifying code of the subtable or subtype</td>
</tr>
<tr>
<td>parent</td>
<td>INTEGER</td>
<td>Identifying code of the supertable or supertype</td>
</tr>
</tbody>
</table>
| class  | CHAR(1) | Inheritance class:  
|        |         | t = named ROW type  
|        |         | T = table |

The child and parent values are from sysxtdtypes.extended_id for named ROW types, or from systables.tabid for tables. Simple indexes on the child and parent columns allow duplicate values.

### SYSLANGAUTH (IDS)

The syslangauth system catalog table contains the authorization information on computer languages that are used to write user-defined routines (UDRs).

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantor</td>
<td>VARCHAR(32)</td>
<td>Name of the grantor of the language authorization</td>
</tr>
<tr>
<td>grantee</td>
<td>VARCHAR(32)</td>
<td>Name of the grantee of the language authorization</td>
</tr>
<tr>
<td>langid</td>
<td>INTEGER</td>
<td>Identifying code of language in sysroutinelangs table</td>
</tr>
</tbody>
</table>
| langauth | CHAR(1)   | The language authorization:  
|          |          | u = Usage privilege granted  
|          |          | U = Usage privilege granted WITH GRANT OPTION |
A composite index on the `langid, grantor,` and `grantee` columns allows only unique values. A composite index on the `langid` and `grantee` columns allows duplicate values.

**SYSLOGMAP (IDS)**

The `syslogmap` system catalog table contains fragmentation information.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabloc</td>
<td>INTEGER</td>
<td>Code for the location of an external table</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Unique identifying code of the table</td>
</tr>
<tr>
<td>fragid</td>
<td>INTEGER</td>
<td>Identifying code of the fragment</td>
</tr>
<tr>
<td>flags</td>
<td>INTEGER</td>
<td>Bitmap of modifiers from declaration of fragment</td>
</tr>
</tbody>
</table>

A simple index on the `tabloc` column and a composite index on the `tabid` and `fragid` columns do not allow duplicate values.

**SYSNEWDEPEND (XPS)**

The `sysnewdepend` system catalog table contains information about generalized-key indexes that are not available in the `sysindexes` table. The dependencies between a generalized-key index and the tables in the FROM clause of the CREATE INDEX statement are stored in the `sysnewdepend` table, which has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>scrid1</td>
<td>VARCHAR(128)</td>
<td>Name of the generalized-key index</td>
</tr>
<tr>
<td>scrid2</td>
<td>INTEGER</td>
<td>Unique identifying code ( = <code>tabid</code>) of the indexed table</td>
</tr>
<tr>
<td>type</td>
<td>INTEGER</td>
<td>Code for the type of generalized-key index</td>
</tr>
<tr>
<td>destid1</td>
<td>INTEGER</td>
<td>The <code>systables.tabid</code> value for the table on which the generalized-key index depends</td>
</tr>
<tr>
<td>destid2</td>
<td>INTEGER</td>
<td>The column number within the <code>destid1</code> table</td>
</tr>
</tbody>
</table>

A composite index on the `scrid1, scrid2,` and `type` columns allows duplicate values. Another composite index on the `destid1, destid2,` and `type` columns also allows duplicate values.

**SYSOBJSTATE (IDS)**

The `sysobjstate` system catalog table stores information about the state (object mode) of database objects. The types of database objects that are listed in this table are indexes, triggers, and constraints.

Every index, trigger, and constraint in the database has a corresponding row in the `sysobjstate` table if a user creates the object. Indexes that the database server creates on the system catalog tables are not listed in the `sysobjstate` table because their object mode cannot be changed.
The **sysobjstate** table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>objtype</td>
<td>CHAR(1)</td>
<td>Code for the type of database object:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C = Constraint</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I = Index</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T = Trigger</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR(32)</td>
<td>Name of the owner of the database object</td>
</tr>
<tr>
<td>name</td>
<td>VARCHAR(128)</td>
<td>Name of the database object</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Identifying code of table on which the object is defined</td>
</tr>
<tr>
<td>state</td>
<td>CHAR(1)</td>
<td>The current state (object mode) of the database object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This value can be one of the following codes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D = Disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E = Enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F = Filtering with no integrity-violation errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>G = Filtering with integrity-violation errors</td>
</tr>
</tbody>
</table>

A composite index on the **objtype**, **name**, **owner**, and **tabid** columns allows only unique combinations of values. A simple index on the **tabid** column allows duplicate values.

**SYSOPCLASSES (IDS)**

The **sysopclasses** system catalog table contains information about operator classes associated with secondary access methods. It contains one row for each operator class that has been defined in the database. The **sysopclasses** table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>opclassname</td>
<td>VARCHAR(128)</td>
<td>Name of the operator class</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR(32)</td>
<td>Name of the owner of the operator class</td>
</tr>
<tr>
<td>amid</td>
<td>INTEGER</td>
<td>Identifying code of the secondary access method associated with this operator class</td>
</tr>
<tr>
<td>opclassid</td>
<td>SERIAL</td>
<td>Identifying code of the operator class</td>
</tr>
<tr>
<td>ops</td>
<td>LVARCHAR</td>
<td>List of names of the operators that belong to this operator class</td>
</tr>
<tr>
<td>support</td>
<td>LVARCHAR</td>
<td>List of names of support functions defined for this operator class</td>
</tr>
</tbody>
</table>

The **opclassid** value corresponds to the **sysams.am_defopclass** value that specifies the default operator class for the secondary access method that the **amid** column specifies.

The **sysopclasses** table has a composite index on the **opclassname** and **owner** columns and an index on **opclassid** column. Both indexes allow only unique values.
The `sysopclstr` system catalog table defines each optical cluster in the database. It contains one row for each optical cluster. The `sysopclstr` table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>VARCHAR(32)</td>
<td>Name of the owner of the optical cluster</td>
</tr>
<tr>
<td>clstrname</td>
<td>VARCHAR(128)</td>
<td>Name of the optical cluster</td>
</tr>
<tr>
<td>clstrsize</td>
<td>INTEGER</td>
<td>Size of the optical cluster</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Unique identifying code for the table</td>
</tr>
<tr>
<td>blobcol1</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 1</td>
</tr>
<tr>
<td>blobcol2</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 2</td>
</tr>
<tr>
<td>blobcol3</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 3</td>
</tr>
<tr>
<td>blobcol4</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 4</td>
</tr>
<tr>
<td>blobcol5</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 5</td>
</tr>
<tr>
<td>blobcol6</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 6</td>
</tr>
<tr>
<td>blobcol7</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 7</td>
</tr>
<tr>
<td>blobcol8</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 8</td>
</tr>
<tr>
<td>blobcol9</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 9</td>
</tr>
<tr>
<td>blobcol10</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 10</td>
</tr>
<tr>
<td>blobcol11</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 11</td>
</tr>
<tr>
<td>blobcol12</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 12</td>
</tr>
<tr>
<td>blobcol13</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 13</td>
</tr>
<tr>
<td>blobcol14</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 14</td>
</tr>
<tr>
<td>blobcol15</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 15</td>
</tr>
<tr>
<td>blobcol16</td>
<td>SMALLINT</td>
<td>BYTE or TEXT column number 16</td>
</tr>
<tr>
<td>clstrkey1</td>
<td>SMALLINT</td>
<td>Cluster key number 1</td>
</tr>
<tr>
<td>clstrkey2</td>
<td>SMALLINT</td>
<td>Cluster key number 2</td>
</tr>
<tr>
<td>clstrkey3</td>
<td>SMALLINT</td>
<td>Cluster key number 3</td>
</tr>
<tr>
<td>clstrkey4</td>
<td>SMALLINT</td>
<td>Cluster key number 4</td>
</tr>
<tr>
<td>clstrkey5</td>
<td>SMALLINT</td>
<td>Cluster key number 5</td>
</tr>
<tr>
<td>clstrkey6</td>
<td>SMALLINT</td>
<td>Cluster key number 6</td>
</tr>
<tr>
<td>clstrkey7</td>
<td>SMALLINT</td>
<td>Cluster key number 7</td>
</tr>
<tr>
<td>clstrkey8</td>
<td>SMALLINT</td>
<td>Cluster key number 8</td>
</tr>
<tr>
<td>clstrkey9</td>
<td>SMALLINT</td>
<td>Cluster key number 9</td>
</tr>
<tr>
<td>clstrkey10</td>
<td>SMALLINT</td>
<td>Cluster key number 10</td>
</tr>
<tr>
<td>clstrkey11</td>
<td>SMALLINT</td>
<td>Cluster key number 11</td>
</tr>
<tr>
<td>clstrkey12</td>
<td>SMALLINT</td>
<td>Cluster key number 12</td>
</tr>
<tr>
<td>clstrkey13</td>
<td>SMALLINT</td>
<td>Cluster key number 13</td>
</tr>
<tr>
<td>clstrkey14</td>
<td>SMALLINT</td>
<td>Cluster key number 14</td>
</tr>
<tr>
<td>clstrkey15</td>
<td>SMALLINT</td>
<td>Cluster key number 15</td>
</tr>
<tr>
<td>clstrkey16</td>
<td>SMALLINT</td>
<td>Cluster key number 16</td>
</tr>
</tbody>
</table>
The contents of this table are sensitive to CREATE OPTICAL CLUSTER, ALTER OPTICAL CLUSTER, and DROP OPTICAL CLUSTER statements that have been executed on databases that support optical cluster subsystems. Changes that affect existing optical clusters are reflected in this table only after you run the UPDATE STATISTICS statement.

A composite index on the clstrname and owner columns allows only unique values. A simple index on the tabid column allows duplicate values.

**SYSPROCAUTH**

The sysprocauth system catalog table describes the privileges granted on a procedure or function. It contains one row for each set of privileges that is granted. The sysprocauth table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantor</td>
<td>VARCHAR(32)</td>
<td>Name of grantor of privileges to access the routine</td>
</tr>
<tr>
<td>grantee</td>
<td>VARCHAR(32)</td>
<td>Name of grantee of privileges to access the routine</td>
</tr>
<tr>
<td>procid</td>
<td>INTEGER</td>
<td>Unique identifying code of the routine</td>
</tr>
<tr>
<td>procauth</td>
<td>CHAR(1)</td>
<td>Type of privilege granted on the routine: e = Execute privilege on routine</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E = Execute privilege WITH GRANT OPTION</td>
</tr>
</tbody>
</table>

A composite index on the procid, grantor, and grantee columns allows only unique values. A composite index on the procid and grantee columns allows duplicate values.

**SYSPROCBODY**

The sysprocbody system catalog table describes the compiled version of each procedure or function in the database. Because the sysprocbody table stores the text of the routine, each routine can have multiple rows. The sysprocbody table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>procid</td>
<td>INTEGER</td>
<td>Unique identifying code for the routine</td>
</tr>
<tr>
<td>datakey</td>
<td>CHAR(1)</td>
<td>Type of information in the data column:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A = Routine alter SQL (will not change this value after update statistics)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D = Routine user documentation text</td>
</tr>
<tr>
<td></td>
<td></td>
<td>E = Time of creation information</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L = Literal value (that is, literal number or quoted string)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P = Interpreter instruction code (p-code)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R = Routine return value type list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S = Routine symbol table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T = Routine text creation SQL</td>
</tr>
<tr>
<td>seqno</td>
<td>INTEGER</td>
<td>Line number within the routine</td>
</tr>
<tr>
<td>data</td>
<td>CHAR(256)</td>
<td>Actual text of the routine</td>
</tr>
</tbody>
</table>

The A flag indicates the procedure modifiers are altered. ALTER ROUTINE statement updates only modifiers and not the routine body. UPDATE STATISTICS updates the query plan and not the routine modifiers, and the value of datakey
will not be changed from A. The A flag gets all the procedures, routines, and functions that were altered with modifiers, including overloaded modifiers. The T flag is used for routine creation text.

The data column contains actual data, which can be in one of these formats:
- Encoded return values list
- Encoded symbol table
- Literal data
- P-code for the routine
- Compiled code for the routine
- Text of the routine and its documentation

A composite index on the procid, datakey, and seqno columns allows only unique values.

**SYSPROCCOLUMNS**

The sysproccolumns system catalog table stores information about return types and parameter names of all UDRs in SYSPROCEDURES.

A composite index on the procid and paramid columns in this table allows only unique values.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>procid</td>
<td>INTEGER</td>
<td>Unique identifying code of the routine</td>
</tr>
<tr>
<td>paramid</td>
<td>INTEGER</td>
<td>Unique identifying code of the parameter</td>
</tr>
<tr>
<td>paramname</td>
<td>VARCHAR (IDENTSIZE)</td>
<td>Name of the parameter</td>
</tr>
<tr>
<td>paramtype</td>
<td>SMALLINT</td>
<td>Identifies the type of parameter</td>
</tr>
<tr>
<td>paramlen</td>
<td>SMALLINT</td>
<td>Specifies the length of the parameter</td>
</tr>
<tr>
<td>pxid</td>
<td>INTEGER</td>
<td>Specifies the extended type ID for the parameter</td>
</tr>
</tbody>
</table>
| paramattr | INTEGER     | 0 = Parameter is of unknown type  
1 = Parameter is INPUT mode  
2 = Parameter is INOUT mode  
3 = Parameter is multiple return value  
4 = Parameter is OUT mode  
5 = Parameter is a return value  |

**SYSPROCEDURES**

The sysprocedures system catalog table lists the characteristics for each function and procedure in the database. It contains one row for each routine.

Each function in sysprocedures has a unique value, procid, called a routine identifier. Throughout the system catalog, a function is identified by its routine identifier, not by its name.
For Extended Parallel Server, `sysprocedures` has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>procname</td>
<td>VARCHAR(128)</td>
<td>Name of routine</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR(32)</td>
<td>Name of owner</td>
</tr>
<tr>
<td>procid</td>
<td>SERIAL</td>
<td>Unique identifying code for the routine</td>
</tr>
<tr>
<td>mode</td>
<td>CHAR(1)</td>
<td>Mode type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D or d = DBA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O or o = Owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P or p = Protected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R or r = Restricted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T or t = Trigger</td>
</tr>
<tr>
<td>retsize</td>
<td>INTEGER</td>
<td>Compiled size (in bytes) of values</td>
</tr>
<tr>
<td>symsize</td>
<td>INTEGER</td>
<td>Compiled size (in bytes) of symbol table</td>
</tr>
<tr>
<td>datasize</td>
<td>INTEGER</td>
<td>Compiled size (in bytes) of constant data</td>
</tr>
<tr>
<td>codesize</td>
<td>INTEGER</td>
<td>Compiled size (in bytes) of routine instruction code</td>
</tr>
<tr>
<td>numargs</td>
<td>INTEGER</td>
<td>Number of arguments to routine</td>
</tr>
</tbody>
</table>

A composite index on `procname` and `owner` requires unique values.

For Dynamic Server `sysprocedures` has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>procname</td>
<td>VARCHAR(128)</td>
<td>Name of routine</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR(32)</td>
<td>Name of owner</td>
</tr>
<tr>
<td>procid</td>
<td>SERIAL</td>
<td>Unique identifying code for the routine</td>
</tr>
<tr>
<td>mode</td>
<td>CHAR(1)</td>
<td>Mode type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D or d = DBA</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O or o = Owner</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P or p = Protected</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R or r = Restricted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T or t = Trigger</td>
</tr>
<tr>
<td>retsize</td>
<td>INTEGER</td>
<td>Compiled size (in bytes) of returned values</td>
</tr>
<tr>
<td>symsize</td>
<td>INTEGER</td>
<td>Compiled size (in bytes) of symbol table</td>
</tr>
<tr>
<td>datasize</td>
<td>INTEGER</td>
<td>Compiled size (in bytes) of constant data</td>
</tr>
<tr>
<td>codesize</td>
<td>INTEGER</td>
<td>Compiled size (in bytes) of routine code</td>
</tr>
<tr>
<td>numargs</td>
<td>INTEGER</td>
<td>Number of arguments to routine</td>
</tr>
<tr>
<td>isproc</td>
<td>CHAR(1)</td>
<td>Whether routine is a procedure or a function:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t = procedure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f = function</td>
</tr>
<tr>
<td>specificname</td>
<td>VARCHAR(128)</td>
<td>Specific name for the routine</td>
</tr>
<tr>
<td>externalname</td>
<td>VARCHAR(255)</td>
<td>Location of the external routine. This item is language-specific in content and format.</td>
</tr>
<tr>
<td>paramstyle</td>
<td>CHAR(1)</td>
<td>Parameter style: I = Informix</td>
</tr>
<tr>
<td>langid</td>
<td>INTEGER</td>
<td>Language code (in <code>sysroutinelangs</code> table)</td>
</tr>
<tr>
<td>paramtypes</td>
<td>RTNPARAMTYPES</td>
<td>Information describing returned parameters</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Explanation</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>variant</td>
<td>BOOLEAN</td>
<td>Whether the routine is VARIANT or not:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(t = \text{is VARIANT})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(f = \text{is not VARIANT})</td>
</tr>
<tr>
<td>client</td>
<td>BOOLEAN</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>handlesnulls</td>
<td>BOOLEAN</td>
<td>NULL handling indicator:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(t = \text{handles NULLs})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(f = \text{does not handle NULLs})</td>
</tr>
<tr>
<td>percallcost</td>
<td>INTEGER</td>
<td>Amount of CPU per call</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Integer cost to execute UDR: cost/call - 0 (-2^{31}-1)</td>
</tr>
<tr>
<td>commutator</td>
<td>VARCHAR(128)</td>
<td>Name of commutator function</td>
</tr>
<tr>
<td>negator</td>
<td>VARCHAR(128)</td>
<td>Name of negator function</td>
</tr>
<tr>
<td>selfunc</td>
<td>VARCHAR(128)</td>
<td>Name of function to estimate selectivity of UDR</td>
</tr>
<tr>
<td>internal</td>
<td>BOOLEAN</td>
<td>Whether the routine can be called from SQL:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(t = \text{routine is internal, not callable from SQL})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(f = \text{routine is external, callable from SQL})</td>
</tr>
<tr>
<td>class</td>
<td>CHAR(18)</td>
<td>CPU class in which routine should be executed</td>
</tr>
<tr>
<td>stack</td>
<td>INTEGER</td>
<td>Stack size in bytes required per invocation</td>
</tr>
<tr>
<td>parallelizable</td>
<td>BOOLEAN</td>
<td>Parallelization indicator for UDR:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(t = \text{parallelizable})</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(f = \text{not parallelizable})</td>
</tr>
<tr>
<td>costfunc</td>
<td>VARCHAR(128)</td>
<td>Name of cost function for UDR</td>
</tr>
<tr>
<td>selconst</td>
<td>SMALLFLOAT</td>
<td>Selectivity constant for UDR</td>
</tr>
</tbody>
</table>

In the mode column, the R mode is a special case of the O mode. A routine is in restricted (R) mode if it was created with a specified owner who is different from the routine creator. If routine statements involving a remote database are executed, the database server uses the permissions of the user who executes the routine instead of the permissions of the routine owner. In all other scenarios, R-mode routines behave the same as O-mode routines.

You cannot use the DROP FUNCTION, DROP ROUTINE, or DROP PROCEDURE statements to delete a protected routine. Protected routines are indicated by lowercase in the mode column. In earlier versions, protected SPL routines (which cannot be deleted) were indicated by a p. Protected SPL routines are treated as DBA routines and cannot be Owner routines. Thus \(D\) and \(O\) indicate DBA and Owner routines, and \(d\) and \(o\) indicate protected DBA and protected Owner routines.

A database server can create protected SPL routines for internal use. These protected SPL routines have \(p\) in the mode column. You cannot modify, drop, or display protected SPL routines.

**Important:** After you issue the SET SESSION AUTHORIZATION statement, the database server assigns a restricted mode to all owner routines that you created while using the new identity.
The database server can create protected routines for internal use. The `sysprocedures` table identifies these protected routines with the letter P or p in the `mode` column. You cannot modify or drop protected routines, nor can you display them through DBSCHEMA.

A unique index is on the `procid` column. A composite index on the `procname`, `isproc`, `numargs`, and `owner` columns allows duplicate values, as does a composite index on the `specificname` and `owner` columns.

### SYSPROCPLAN

The `sysprocplan` system catalog table describes the query-execution plans and dependency lists for data-manipulation statements within each routine. Because different parts of a routine plan can be created on different dates, this table can contain multiple rows for each routine.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>procid</td>
<td>INTEGER</td>
<td>Identifying code for the routine</td>
</tr>
<tr>
<td>planid</td>
<td>INTEGER</td>
<td>Identifying code for the plan</td>
</tr>
<tr>
<td>datakey</td>
<td>CHAR(1)</td>
<td>Type of information stored in data column: D = Dependency list</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I = Information record</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q = Execution plan</td>
</tr>
<tr>
<td>seqno</td>
<td>INTEGER</td>
<td>Line number within the plan</td>
</tr>
<tr>
<td>created</td>
<td>DATE</td>
<td>Date when plan was created</td>
</tr>
<tr>
<td>data</td>
<td>CHAR(256)</td>
<td>Encoded (compiled) list or plan (IDS)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Text of the SPL routine (XPS)</td>
</tr>
<tr>
<td>collation</td>
<td>CHAR(32)</td>
<td>Collating order at the time when routine was created</td>
</tr>
</tbody>
</table>

Before a routine is run, its dependency list in the `data` column is examined. If the major version number of a table accessed by the plan has changed, or if any object that the routine uses has been modified since the plan was optimized (for example, if an index has been dropped), then the plan is optimized again. When `datakey` is I, the `data` column stores information about UPDATE STATISTICS and PDQPRINTER.

It is possible to delete all the plans for a given routine by using the DELETE statement on `sysprocplan`. When the routine is subsequently executed, new plans are automatically generated and recorded in `sysprocplan`. The UPDATE STATISTICS FOR PROCEDURE statement also updates this table.

A composite index on the `procid`, `planid`, `datakey`, and `seqno` columns allows only unique values.
**SYSREFERENCES**

The `sysreferences` system catalog table lists all referential constraints on columns. It contains a row for each referential constraint in the database.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>constrid</td>
<td>INTEGER</td>
<td>Code uniquely identifying the constraint</td>
</tr>
<tr>
<td>primary</td>
<td>INTEGER</td>
<td>Identifying code of the corresponding primary key</td>
</tr>
<tr>
<td>ptabid</td>
<td>INTEGER</td>
<td>Identifying code of the table that is the primary key</td>
</tr>
<tr>
<td>updrule</td>
<td>CHAR(1)</td>
<td>Reserved for future use; displays an R</td>
</tr>
<tr>
<td>delrule</td>
<td>CHAR(1)</td>
<td>Whether constraint uses cascading delete or restrict rule: C = Cascading delete R = Restrict (default)</td>
</tr>
<tr>
<td>matchtype</td>
<td>CHAR(1)</td>
<td>Reserved for future use; displays an N</td>
</tr>
<tr>
<td>pendant</td>
<td>CHAR(1)</td>
<td>Reserved for future use; displays an N</td>
</tr>
</tbody>
</table>

The `constrid` column is indexed and allows only unique values. The `primary` column is indexed and allows duplicate values.

**SYSREPOSITORY (XPS)**

The `sysrepository` system catalog table contains data about generalized-key indexes that the `sysindexes` system catalog table does not provide.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>VARCHAR(128)</td>
<td>Index from the generalized-key (GK) index</td>
</tr>
<tr>
<td>id2</td>
<td>INTEGER</td>
<td>Tabid of table with the generalized-key index</td>
</tr>
<tr>
<td>type</td>
<td>INTEGER</td>
<td>Integer code for type of object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>In this release, the only value that can appear is 1, indicating a GK index type.</td>
</tr>
<tr>
<td>seqid</td>
<td>SERIAL</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(This value is not related to <code>syssequences.seqid</code>.)</td>
</tr>
<tr>
<td>desc</td>
<td>TEXT</td>
<td>The CREATE INDEX statement of a GK index</td>
</tr>
<tr>
<td>bin</td>
<td>BYTE</td>
<td>Internal representation of the generalized-key index</td>
</tr>
</tbody>
</table>

The contents of the `sysrepository` table are useful when a generalized-key index has to be rebuilt during a recovery or if a user wants to see the CREATE statement for a specific generalized-key index.

The `desc` column contains the CREATE statement for each generalized-key index in the database.

An index on the `seqid` column allows duplicate values. A composite index on the `id1`, `id2`, and `type` columns requires unique combinations of values.
SYSROLEAUTH

The `sysroleauth` system catalog table describes the roles that are granted to users. It contains one row for each role that is granted to a user in the database. The `sysroleauth` table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>rolename</td>
<td>VARCHAR(32)</td>
<td>Name of the role</td>
</tr>
<tr>
<td>grantee</td>
<td>VARCHAR(32)</td>
<td>Name of the grantee of the role</td>
</tr>
<tr>
<td>is_grantable</td>
<td>CHAR(1)</td>
<td>Specifies whether the role is grantable:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y = Grantable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N = Not grantable</td>
</tr>
</tbody>
</table>

The `is_grantable` column indicates whether the role was granted with the WITH GRANT OPTION of the GRANT statement.

A composite index on the `rolename` and `grantee` columns allows only unique values.

SYSROUTINELANGS (IDS)

The `sysroutinelangs` system catalog table lists the supported programming languages for user-defined routines (UDRs). It has these columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>langid</td>
<td>SERIAL</td>
<td>Code uniquely identifying a supported language</td>
</tr>
<tr>
<td>langname</td>
<td>CHAR(30)</td>
<td>Name of the language, such as C or SPL</td>
</tr>
<tr>
<td>langinitfunc</td>
<td>VARCHAR(128)</td>
<td>Name of initialization function for the language</td>
</tr>
<tr>
<td>langpath</td>
<td>CHAR(255)</td>
<td>Directory path for the UDR language</td>
</tr>
<tr>
<td>langclass</td>
<td>CHAR(18)</td>
<td>Name of the class of the UDR language</td>
</tr>
</tbody>
</table>

An index on the `langname` column allows duplicate values.

SYSSECLABELCOMPONENTS

The `sysseclabelcomponents` system catalog table records security label components. It has these columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>compname</td>
<td>VARCHAR(128)</td>
<td>Component name</td>
</tr>
<tr>
<td>compid</td>
<td>SERIAL</td>
<td>Component ID</td>
</tr>
<tr>
<td>comptype</td>
<td>CHAR(1)</td>
<td>The component type:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A = array</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S = set</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T = tree</td>
</tr>
<tr>
<td>numelements</td>
<td>INTEGER</td>
<td>Number of elements in the component</td>
</tr>
<tr>
<td>coveringinfo</td>
<td>VARCHAR(128)</td>
<td>Internal encoding information</td>
</tr>
<tr>
<td>numalters</td>
<td>SMALLINT</td>
<td>Numbers of alter operations that have been performed on the component</td>
</tr>
</tbody>
</table>
**SYSSECLABELCOMPONENTELEMENTS**

The **sysseclabelcomponentelements** system catalog table records the values of component elements of security labels. It has these columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>compid</td>
<td>INTEGER</td>
<td>Component ID</td>
</tr>
<tr>
<td>element</td>
<td>VARCHAR(32)</td>
<td>Element name</td>
</tr>
<tr>
<td>elementencoding</td>
<td>CHAR(8)</td>
<td>Encoded form of the element</td>
</tr>
</tbody>
</table>
| parentelement  | VARCHAR(32)| The name of the parent elements for tree components. The value is NULL for the following items:  
Set components  
Array components  
Root nodes of a tree component |
| alterversion   | SMALLINT   | The number of the alter operation when the element is added. This value is used by the dbexport and dbimport commands. |

**SYSSECPOLICIES**

The **syssecpolicies** system catalog table records security policies. It has these columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>secpolicyname</td>
<td>VARCHAR(128)</td>
<td>Security policy name</td>
</tr>
<tr>
<td>secpolicyid</td>
<td>SERIAL</td>
<td>Security policy ID</td>
</tr>
<tr>
<td>numcomps</td>
<td>SMALLINT</td>
<td>Number of security label components in the security policy</td>
</tr>
</tbody>
</table>
| comptypelist   | CHAR(16)   | An ordered list of the type of each component in the policy.  
A = array  
S = set  
T = tree  
-= Beyond NUMCOMPS |
| overrideseclabel| CHAR(1)   | Indicates the behavior when a user's security label and exemption credentials do not allow them to insert or update a data row with the security that is label provided on the INSERT or UPDATE SQL statement.  
• Y: The security label provided is ignored and replaced by the user's security label for write access.  
• N: Return an error when not authorized to write a security label. |
SYSSECPOLOICYCOMPONENTS

The syssecpolicycomponents system catalog table records the components for each security policies. It has these columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>secpolicyid</td>
<td>INTEGER</td>
<td>Security policy ID</td>
</tr>
<tr>
<td>compid</td>
<td>INTEGER</td>
<td>ID of a component of the label security policy</td>
</tr>
<tr>
<td>compno</td>
<td>SMALLINT</td>
<td>Position of the security label component as it appears in the security policy, starting with position 1.</td>
</tr>
</tbody>
</table>

SYSSECPOLOICYEXEMPTIONS

The syssecpolicyexemptions system catalog table records the exemptions that have been given to users. It has these columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantee</td>
<td>CHAR(32)</td>
<td>The user who has this exemption</td>
</tr>
<tr>
<td>secpolicyid</td>
<td>INTEGER</td>
<td>ID of the policy on which the exemption is granted</td>
</tr>
</tbody>
</table>
| exemption | CHAR(6)  | The exemption given to the user who is identified in the GRANTEE column. The six characters have the following meanings:

1 = Read array
2 = Read set
3 = Read tree
4 = Write array
5 = Write set
6 = Write tree

Each character has one of the following values:

E = Exempt
D = Write down exemption
U = Write up exemption
– = No exemption |

SYSSECLABELS

The sysseclabels system catalog table records the security label encoding. It has these columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>secpolicyid</td>
<td>INTEGER</td>
<td>ID of the security policy to which the security label belongs</td>
</tr>
<tr>
<td>seclabelid</td>
<td>INTEGER</td>
<td>Security label ID</td>
</tr>
<tr>
<td>sysseclabelnames</td>
<td>VARCHAR(128)</td>
<td>Security label encoding</td>
</tr>
</tbody>
</table>
SYSSECLABELNAMES

The **sysseclabelnames** system catalog table records the security label names. It has these columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>secpolicyid</td>
<td>INTEGER</td>
<td>The ID of the security policy to which the security label belongs.</td>
</tr>
<tr>
<td>seclabelname</td>
<td>VARCHAR(128)</td>
<td>The name of the security label</td>
</tr>
<tr>
<td>seclabelid</td>
<td>INTEGER</td>
<td>The ID of the security label</td>
</tr>
</tbody>
</table>

SYSSECLABELAUTH

The **sysseclabelauth** system catalog table records the labels that have been granted to users. It has these columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRANTEE</td>
<td>CHAR(32)</td>
<td>The name of the label grantee</td>
</tr>
<tr>
<td>secpolicyid</td>
<td>INTEGER</td>
<td>The ID of the security policy to which the security label belongs.</td>
</tr>
<tr>
<td>readseclabelid</td>
<td>INTEGER</td>
<td>The security label ID of the security label granted for read access</td>
</tr>
<tr>
<td>writeseclabelid</td>
<td>INTEGER</td>
<td>The security label ID of the security label granted for write access</td>
</tr>
</tbody>
</table>

SYSSEQUENCES (IDS)

The **syssequences** system catalog table lists the sequence objects that exist in the database. The **syssequences** table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>seqid</td>
<td>SERIAL</td>
<td>Code uniquely identifying the sequence object</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Identifying code of the sequence as a table object</td>
</tr>
<tr>
<td>start_val</td>
<td>INT8</td>
<td>Starting value of the sequence</td>
</tr>
<tr>
<td>inc_val</td>
<td>INT8</td>
<td>Value of the increment between successive values</td>
</tr>
<tr>
<td>max_val</td>
<td>INT8</td>
<td>Largest possible value of the sequence</td>
</tr>
<tr>
<td>min_val</td>
<td>INT8</td>
<td>Smallest possible value of the sequence</td>
</tr>
<tr>
<td>cycle</td>
<td>CHAR(1)</td>
<td>Zero means NOCYCLE, 1 means CYCLE</td>
</tr>
<tr>
<td>cache</td>
<td>INTEGER</td>
<td>Number of preallocated values in sequence cache</td>
</tr>
<tr>
<td>order</td>
<td>CHAR(1)</td>
<td>Zero means NOORDER, 1 means ORDER</td>
</tr>
</tbody>
</table>

SYSSYNONYMS

The **syssynonyms** system catalog table lists the synonyms for each table or view. Except for database servers that have migrated from certain interim releases of Version 1.10 Informix database servers, only the **syssyntable** table describes synonyms, and the **syssynonyms** table is unused. It has the following columns.
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>VARCHAR(32)</td>
<td>Name of the owner of the synonym</td>
</tr>
<tr>
<td>synname</td>
<td>VARCHAR(128)</td>
<td>Name of the synonym</td>
</tr>
<tr>
<td>created</td>
<td>DATE</td>
<td>Date when the synonym was created</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Identifying code of a table, sequence, or view</td>
</tr>
</tbody>
</table>

A composite index on the **owner** and **synonym** columns allows only unique values. The **tabid** column is indexed and allows duplicate values.

**SYSSYNTABLE**

The **syssyntable** system catalog table outlines the mapping between each public or private synonym and the database object (table, sequence, or view) that it represents. It contains one row for each entry in the **systables** table that has a **tabtype** value of P or S. The **syssyntable** table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Identifying code of the public synonym</td>
</tr>
<tr>
<td>servername</td>
<td>VARCHAR(128)</td>
<td>Name of an external database server</td>
</tr>
<tr>
<td>dbname</td>
<td>VARCHAR(128)</td>
<td>Name of an external database</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR(32)</td>
<td>Name of the owner of an external object</td>
</tr>
<tr>
<td>tabname</td>
<td>VARCHAR(128)</td>
<td>Name of an external table or view</td>
</tr>
<tr>
<td>btabid</td>
<td>INTEGER</td>
<td>Identifying code of a base table, sequence, or view</td>
</tr>
</tbody>
</table>

ANSI-compliant databases do not support public synonyms; their **syssyntable** tables can describe only synonyms whose **syssyntable.tabtype** value is P.

If you define a synonym for an object that is in your current database, only the **tabid** and **btabid** columns are used. If you define a synonym for a table that is external to your current database, the **btabid** column is not used, but the **tabid**, **servername**, **dbname**, **owner**, and **tabname** columns are used.

The **tabid** column maps to **systables.tabid**. With the **tabid** information, you can determine additional facts about the synonym from **systables**.

An index on the **tabid** column allows only unique values. The **btabid** column is indexed to allow duplicate values.

**SYSTABAMDATA (IDS)**

The **systabamdata** system catalog table stores the table-specific hashing parameters of tables that were created with a primary access method.

The **systabamdata** table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Identifying code of the table</td>
</tr>
<tr>
<td>am_param</td>
<td>CHAR(256)</td>
<td>Access method parameter choices</td>
</tr>
<tr>
<td>am_space</td>
<td>VARCHAR(128)</td>
<td>Name of the storage space holding the data values</td>
</tr>
</tbody>
</table>
The **am_param** column stores configuration parameters that determine how a primary access method accesses a given table. Each configuration parameter in the **am_param** list has the format *keyword=value* or *keyword*.

The **am_space** column specifies the location of the table. It might reside in a cooked file, a different database, or an sbspace within the database server.

The **tabid** column is the primary key to the **systables** table. This column is indexed and must contain unique values.

---

**SYSTABAUTH**

The **systabauth** system catalog table describes each set of privileges that are granted on a table, view, sequence, or synonym. It contains one row for each set of table privileges that are granted in the database; the REVOKE statement can modify a row. The **systabauth** table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantor</td>
<td>VARCHAR(32)</td>
<td>Name of the grantor of privilege</td>
</tr>
<tr>
<td>grantee</td>
<td>VARCHAR(32)</td>
<td>Name of the grantee of privilege</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Value from <strong>systables.tabid</strong> for database object</td>
</tr>
<tr>
<td>tabauth</td>
<td>CHAR(9)</td>
<td>Pattern that specifies privileges on the table, view, synonym, or (IDS) sequence:</td>
</tr>
<tr>
<td></td>
<td>CHAR(8)</td>
<td>s or S = Select</td>
</tr>
<tr>
<td></td>
<td></td>
<td>u or U = Update</td>
</tr>
<tr>
<td></td>
<td></td>
<td>* = Column-level privilege</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i or I = Insert</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d or D = Delete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x or X = Index</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a or A = Alter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>r or R = References</td>
</tr>
<tr>
<td></td>
<td></td>
<td>n or N = Under privilege (IDS)</td>
</tr>
</tbody>
</table>

If the **tabauth** column shows a privilege code in uppercase (for example, S for Select), this indicates that the user also has the option to grant that privilege to others. Privilege codes listed in lowercase (for example, s for select) indicate that the user has the specified privilege, but cannot grant it to others.

A hyphen ( - ) indicates the absence of the privilege corresponding to that position within the **tabauth** pattern.

A **tabauth** value with an asterisk ( * ) means column-level privileges exist; see also **syscolauth** (page 1-30). (In DB–Access, the **Privileges** option of the **Info** command for a specified table can display the column-level privileges on that table.)

A composite index on **tabid**, **grantor**, and **grantee** allows only unique values. A composite index on **tabid** and **grantee** allows duplicate values.
The **systables** system catalog table contains a row for each table object (a table, view, synonym, or in Dynamic Server, a sequence) that has been defined in the database, including the tables and views of the system catalog.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabname</td>
<td>VARCHAR(128)</td>
<td>Name of table, view, synonym, or (for IDS) sequence</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR(32)</td>
<td>Owner of table (user <strong>informix</strong> for system catalog tables and <strong>username</strong> for database tables)</td>
</tr>
<tr>
<td>partnum</td>
<td>INTEGER</td>
<td>Physical location code</td>
</tr>
<tr>
<td>tabid</td>
<td>SERIAL</td>
<td>System-assigned sequential identifying number</td>
</tr>
<tr>
<td>rowsize</td>
<td>SMALLINT</td>
<td>Row size</td>
</tr>
<tr>
<td>ncols</td>
<td>SMALLINT</td>
<td>Number of columns in the table</td>
</tr>
<tr>
<td>nindexes</td>
<td>SMALLINT</td>
<td>Number of indexes on the table</td>
</tr>
<tr>
<td>nrows</td>
<td>INTEGER</td>
<td>Number of rows in the table</td>
</tr>
<tr>
<td>created</td>
<td>DATE</td>
<td>Date when the table was created or modified</td>
</tr>
<tr>
<td>version</td>
<td>INTEGER</td>
<td>Number that changes when table is altered</td>
</tr>
<tr>
<td>tabtype</td>
<td>CHAR(1)</td>
<td>Code indicating the type of object: T = Table E = External Table V = View Q = Sequence (IDS) P = Private synonym S = Public synonym (Type S is not available in an ANSI-compliant database.)</td>
</tr>
<tr>
<td>locklevel</td>
<td>CHAR(1)</td>
<td>Lock mode for the table: B = Page P = Page R = Row T = Table (XPS)</td>
</tr>
<tr>
<td>npused</td>
<td>INTEGER</td>
<td>Number of data pages that have ever been initialized in the tablespace by the database server</td>
</tr>
<tr>
<td>fextsize</td>
<td>INTEGER</td>
<td>Size of initial extent (in kilobytes)</td>
</tr>
<tr>
<td>nextsize</td>
<td>INTEGER</td>
<td>Size of all subsequent extents (in kilobytes)</td>
</tr>
<tr>
<td>flags</td>
<td>SMALLINT</td>
<td>Codes for classifying permanent tables: ST_RAW ( = 0x00000010 ) (IDS) RAW ( = 0x00000002 ) (XPS) STATIC ( = 0x00000004 ) (XPS) OPERATIONAL ( = 0x00000008 ) (XPS) STANDARD ( = 0x00000010 ) (XPS) EXTERNAL ( = 0x00000020 ) (XPS)</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Explanation</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>site</td>
<td>VARCHAR(128)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>dbname</td>
<td>VARCHAR(128)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>type_xid</td>
<td>INTEGER</td>
<td>Code from sysxtdtypes.extended_id for typed tables, or 0 for untyped tables</td>
</tr>
<tr>
<td>am_id</td>
<td>INTEGER</td>
<td>Access method code (key to sysams table)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NULL or 0 indicates built-in storage manager</td>
</tr>
<tr>
<td>minrowsize</td>
<td>SMALLINT</td>
<td>Minimum row size</td>
</tr>
<tr>
<td>ustatlow</td>
<td>YEAR TO FRACTION(5)</td>
<td>Date and time of the last USTAT LOW run</td>
</tr>
<tr>
<td>secpolicyid</td>
<td>INTEGER</td>
<td>ID of the SECURITY policy attached to the table. NULL for non protected tables</td>
</tr>
<tr>
<td>protgranularity</td>
<td>CHAR(1)</td>
<td>LBAC granularity level:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• R: Row level granularity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• C: Column level granularity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• B: Both column and row granularity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Blank for non-protected tables</td>
</tr>
</tbody>
</table>

Each table, view, sequence, and synonym recorded in the systables table is assigned a tabid, which is a system-assigned SERIAL value that uniquely identifies the object. The first 99 tabid values are reserved for the system catalog. The tabid of the first user-defined table object in a database is always 100.

The tabid column is indexed and contains only unique values. A composite index on the tabname and owner columns also requires unique values.

The version column contains an encoded number that is stored in systables when a new table is created. Portions of this value are incremented when data-definition statements, such as ALTER INDEX, ALTER TABLE, DROP INDEX, and CREATE INDEX, are performed on the table.

In the flags column, ST_RAW represents a nonlogging permanent table in a database that supports transaction logging.

When a prepared statement that references a database table is executed, the version value is checked to make sure that nothing has changed since the statement was prepared. If the version value has been changed by DDL operations that modified the table schema while automatic recompilation was disabled by the IFX_AUTO_REPREPARE setting, the prepared statement is not executed, and you must prepare the statement again.

The npused column does not reflect the number of pages used for BYTE or TEXT data, nor the number of pages that are freed in DELETE or TRUNCATE operations.
The systables table has two rows that store information about the database locale: GL_COLLATE with a tabid of 90 and GL_CTYPE with a tabid of 91. To view these rows, enter the following SELECT statement:

```
SELECT * FROM systables WHERE tabid=90 OR tabid=91
```

### SYSTRACECLASSES (IDS)

The systraceclasses system catalog table contains the names and identifiers of trace classes. The systraceclasses table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>CHAR(18)</td>
<td>Name of the class of trace messages</td>
</tr>
<tr>
<td>classid</td>
<td>SERIAL</td>
<td>Identifying code of the trace class</td>
</tr>
</tbody>
</table>

A trace class is a category of trace messages that you can use in the development and testing of new DataBlade modules and user-defined routines. Developers use the tracing facility by calling the appropriate DataBlade API routines within their code.

To create a new trace class, insert a row directly into the systraceclasses table. By default, all users can view this table, but only users with the DBA privilege can modify it.

The database cannot support tracing unless the MITRACE_OFF configuration parameter is undefined.

A unique index on the name column requires each trace class to have a unique name. The database server assigns to each class a unique sequential code. The index on this classid column also allows only unique values.

### SYSTRACEMSGS (IDS)

The systracemsgs system catalog table stores internationalized trace messages that you can use in debugging user-defined routines.

The systracemsgs table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>VARCHAR(128)</td>
<td>Name of the message</td>
</tr>
<tr>
<td>msgid</td>
<td>SERIAL</td>
<td>Identifying code of the message template</td>
</tr>
<tr>
<td>locale</td>
<td>CHAR(36)</td>
<td>Locale with which this version of the message is associated (for example, en_us.8859-1)</td>
</tr>
<tr>
<td>seqno</td>
<td>SMALLINT</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>message</td>
<td>VARCHAR(255)</td>
<td>The message text</td>
</tr>
</tbody>
</table>

DataBlade module developers create a trace message by inserting a row directly into the systracemsgs table. Once a message is created, the development team can specify it either by name or by msgid code, using trace statements that the DataBlade API provides.
To create a trace message, you must specify its name, locale, and text. By default, all users can view the `sysracemsgs` table, but only users with the DBA privilege can modify it.

The database cannot support tracing unless the MITRACE_OFF configuration parameter is undefined.

A unique composite index is defined on the `name` and `locale` columns. Another unique index is defined on the `msgid` column.

### SYSTRIGBODY

The `sysrigbody` system catalog table contains the ASCII text of the trigger definition and the linearized code for the trigger. *Linearized code* is binary data and code that is represented in ASCII format.

**Important:** The database server uses the linearized code that is stored in `sysrigbody`. You must not alter the content of rows that contain linearized code.

The `sysrigbody` table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>trigid</td>
<td>INTEGER</td>
<td>Identifying code of the trigger</td>
</tr>
<tr>
<td>datakey</td>
<td>CHAR(1)</td>
<td>Code specifying the type of data:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A = ASCII text for the body, triggered actions</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B = Linearized code for the body</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D = English text for the header, trigger definition</td>
</tr>
<tr>
<td></td>
<td></td>
<td>H = Linearized code for the header</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S = Linearized code for the symbol table</td>
</tr>
<tr>
<td>seqno</td>
<td>INTEGER</td>
<td>Page number of this data segment</td>
</tr>
<tr>
<td>data</td>
<td>CHAR(256)</td>
<td>English text or linearized code</td>
</tr>
<tr>
<td>collation</td>
<td>CHAR(32)</td>
<td>Collating order at the time when trigger was created</td>
</tr>
</tbody>
</table>

A composite index on the `trigid`, `datakey`, and `seqno` columns allows only unique values.

### SYSTRIGGERS

The `sysriggers` system catalog table contains information about the SQL triggers in the database. This information includes the triggering event and the correlated reference specification for the trigger. The `sysriggers` table has the following columns.
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>trigid</td>
<td>SERIAL</td>
<td>Identifying code of the trigger</td>
</tr>
<tr>
<td>trigname</td>
<td>VARCHAR(128)</td>
<td>Name of the trigger</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR(32)</td>
<td>Name of the owner of the trigger</td>
</tr>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Identifying code of the triggering table</td>
</tr>
<tr>
<td>event</td>
<td>CHAR(1)</td>
<td>Code for the type of triggering event:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D = Delete trigger</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I = Insert trigger</td>
</tr>
<tr>
<td></td>
<td></td>
<td>U = Update trigger</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S = Select trigger</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d = INSTEAD OF Delete trigger</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i = INSTEAD OF Insert trigger</td>
</tr>
<tr>
<td></td>
<td></td>
<td>u = INSTEAD OF Update trigger (IDS)</td>
</tr>
<tr>
<td>old</td>
<td>VARCHAR(128)</td>
<td>Name of value before update</td>
</tr>
<tr>
<td>new</td>
<td>VARCHAR(128)</td>
<td>Name of value after update</td>
</tr>
<tr>
<td>mode</td>
<td>CHAR(1)</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>

A composite index on the trigname and owner columns allows only unique values. An index on the trigid column also requires unique values. An index on the tabid column allows duplicate values.

**SYSUSERS**

The sysusers system catalog table describes each set of privileges that are granted on the database. It contains one row for each user or role that has privileges on the database. This system catalog table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>username</td>
<td>VARCHAR(32)</td>
<td>Name of the database user or role</td>
</tr>
<tr>
<td>usertype</td>
<td>CHAR(1)</td>
<td>Code specifying database-level privileges:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C = Connect (work within existing tables)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D = DBA (all privileges)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>G = Role</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R = Resource (create permanent tables, user-defined data types, and indexes)</td>
</tr>
<tr>
<td>priority</td>
<td>SMALLINT</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>password</td>
<td>CHAR(16)</td>
<td>Reserved for future use</td>
</tr>
<tr>
<td>defrole</td>
<td>VARCHAR(32)</td>
<td>Name of the default role</td>
</tr>
</tbody>
</table>

An index on username allows only unique values. The username value can be the login name of a user or the name of a role.
SYSVIEWS

The sysviews system catalog table describes each view in the database. Because it stores the SELECT statement that created the view, sysviews can contain multiple rows for each view. It has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabid</td>
<td>INTEGER</td>
<td>Identifying code of the view</td>
</tr>
<tr>
<td>seqno</td>
<td>SMALLINT</td>
<td>Line number of the SELECT statement</td>
</tr>
<tr>
<td>viewtext</td>
<td>CHAR(64)</td>
<td>Actual SELECT statement used to create the view</td>
</tr>
</tbody>
</table>

A composite index on tabid and seqno allows only unique values.

SYSVIOLATIONS

The sysviolations system catalog table stores information about the constraint violations for base tables. Every table in the database that has a violations table and a diagnostics table associated with it has a corresponding row in the sysviolations table, which has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>targettid</td>
<td>INTEGER</td>
<td>Identifying code of the target table (the base table on which the violations table and the diagnostic table are defined)</td>
</tr>
<tr>
<td>viotid</td>
<td>INTEGER</td>
<td>Identifying code of the violations table</td>
</tr>
<tr>
<td>diatid</td>
<td>INTEGER</td>
<td>Identifying code of the diagnostics table</td>
</tr>
<tr>
<td>maxrows</td>
<td>INTEGER</td>
<td>Maximum number of rows that can be inserted into the diagnostics table by a single insert, update, or delete operation on a target table that has a filtering mode object defined on it (IDS) The maximum number of rows allowed in the violations table for each coserver (XPS)</td>
</tr>
</tbody>
</table>

The maxrows column also signifies the maximum number of rows that can be inserted in the diagnostics table during a single operation that enables a disabled object or that sets a disabled object to filtering mode (provided that a diagnostics table exists for the target table). If no maximum is specified for the diagnostics or violations table, then maxrows contains a NULL value.

Extended Parallel Server does not use the diagnostic table when a constraint violation occurs. Rather, the database server stores additional information in the violations table. The violations table contains the data that the transaction refused and an indication of the cause.

The primary key of this table is the targettid column. An additional unique index is also defined on the viotid column.

Dynamic Server also has a unique index on the diatid column.
SYSXADATASOURCES

The sysxadatasources system catalog table stores XA data sources. The sysxadatasources table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>xa_datasrc_owner</td>
<td>CHAR(32)</td>
<td>The user ID of the XA data source owner</td>
</tr>
<tr>
<td>xa_datasrc_name</td>
<td>VARCHAR(128)</td>
<td>The name of the XA data source</td>
</tr>
<tr>
<td>xa_datasrc_rmid</td>
<td>SERIAL</td>
<td>Unique RMID of the XA data source</td>
</tr>
<tr>
<td>xa_source_typeid</td>
<td>INTEGER</td>
<td>XA data source type ID</td>
</tr>
</tbody>
</table>

SYSXASOURCETYPES

The sysxasourcetypes system catalog table stores XA data source types. The sysxasourcetypes table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>xa_source_typeid</td>
<td>SERIAL</td>
<td>A unique identifier for the source type</td>
</tr>
<tr>
<td>xa_source_owner</td>
<td>CHAR(32)</td>
<td>The user ID of the owner</td>
</tr>
<tr>
<td>xa_source_name</td>
<td>VARCHAR(128)</td>
<td>The name of the source type</td>
</tr>
<tr>
<td>xa_flags</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>xa_version</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>xa_open</td>
<td>INTEGER</td>
<td>UDR ID of xa_open_entry</td>
</tr>
<tr>
<td>xa_close</td>
<td>INTEGER</td>
<td>UDR ID of xa_close_entry</td>
</tr>
<tr>
<td>xa_end</td>
<td>INTEGER</td>
<td>UDR ID of xa_end_entry</td>
</tr>
<tr>
<td>xa_rollback</td>
<td>INTEGER</td>
<td>UDR ID of xa_rollback_entry</td>
</tr>
<tr>
<td>xa_prepare</td>
<td>INTEGER</td>
<td>UDR ID of xa_prepare_entry</td>
</tr>
<tr>
<td>xa_commit</td>
<td>INTEGER</td>
<td>UDR ID of xa_commit_entry</td>
</tr>
<tr>
<td>xa_recover</td>
<td>INTEGER</td>
<td>UDR ID of xa_recover_entry</td>
</tr>
<tr>
<td>xa_forget</td>
<td>INTEGER</td>
<td>UDR ID of xa_forget_entry</td>
</tr>
<tr>
<td>xa_complete</td>
<td>INTEGER</td>
<td>UDR ID of xa_complete_entry</td>
</tr>
</tbody>
</table>

SYSXTDDDESC (IDS)

The sysxtddesc system catalog table provides a text description of each UDT defined in the database. The sysxtddesc table has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>extended_id</td>
<td>INTEGER</td>
<td>Code uniquely identifying the extended data types</td>
</tr>
<tr>
<td>seqno</td>
<td>SMALLINT</td>
<td>Value to order and identify one line of the description of the UDT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A new line is created only if the remaining text string is larger than 255 bytes.</td>
</tr>
<tr>
<td>description</td>
<td>CHAR(256)</td>
<td>Textual description of the extended data type</td>
</tr>
</tbody>
</table>

A composite index on extended_id and seqno allows duplicate values.
SYSXTDTYPEAUTH (IDS)

The `sysxtdtypeauth` system catalog table identifies the privileges for each UDT (user-defined data type). The `sysxtdtypeauth` table contains one row for each set of privileges granted and has the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>grantor</td>
<td>VARCHAR(32)</td>
<td>Name of grantor of privilege</td>
</tr>
<tr>
<td>grantee</td>
<td>VARCHAR(32)</td>
<td>Name of grantee of privilege</td>
</tr>
<tr>
<td>type</td>
<td>INTEGER</td>
<td>Code identifying the UDT</td>
</tr>
<tr>
<td>auth</td>
<td>CHAR(2)</td>
<td>Code identifying privileges on the UDT:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>n</code> or <code>N</code> = Under privilege</td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>u</code> or <code>U</code> = Usage privilege</td>
</tr>
</tbody>
</table>

If the privilege code in the auth column is upper case (for example, ‘U’ for usage), a user who has this privilege can also grant it to others. If the code is in lower case, a user who has the privilege cannot grant it to others.

A composite index on type, grantor, and grantee allows only unique values. A composite index on the type and grantee columns allows duplicate values.

SYSXTDTYPES (IDS)

The `sysxtdtype` system catalog table has an entry for each UDT (user-defined data type), including opaque and distinct data types and complex data types (named ROW type, unnamed ROW type, and COLLECTION type), that is defined in the database. The `sysxtdtypes` table has the following columns.
<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>extended_id</td>
<td>SERIAL</td>
<td>Unique identifying code for extended data type</td>
</tr>
<tr>
<td>domain</td>
<td>CHAR(1)</td>
<td>Code for the domain of the UDT</td>
</tr>
<tr>
<td>mode</td>
<td>CHAR(1)</td>
<td>Code classifying the UDT:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>B = Base (opaque) type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>C = Collection type or unnamed ROW type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D = Distinct type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R = Named ROW type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>' ' (blank) = Built-in type</td>
</tr>
<tr>
<td>owner</td>
<td>VARCHAR(32)</td>
<td>Name of the owner of the UDT</td>
</tr>
<tr>
<td>name</td>
<td>VARCHAR(128)</td>
<td>Name of the UDT</td>
</tr>
<tr>
<td>type</td>
<td>SMALLINT</td>
<td>Code classifying the UDT</td>
</tr>
<tr>
<td>source</td>
<td>INTEGER</td>
<td>The sysxtdtypes reference (for distinct types only)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero (0) indicates that a distinct UDT was created from a built-in data type.</td>
</tr>
<tr>
<td>maxlen</td>
<td>INTEGER</td>
<td>The maximum length for variable-length data types</td>
</tr>
<tr>
<td>length</td>
<td>INTEGER</td>
<td>The length in bytes for fixed-length data types</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero indicates a variable-length UDT.</td>
</tr>
<tr>
<td>byvalue</td>
<td>CHAR(1)</td>
<td>'T' = UDT is passed by value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'F' = UDT is not passed by value</td>
</tr>
<tr>
<td>cannothash</td>
<td>CHAR(1)</td>
<td>'T' = UDT is hashable by default hash function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'F' = UDT is not hashable by default function</td>
</tr>
<tr>
<td>align</td>
<td>SMALLINT</td>
<td>Alignment ( = 1, 2, 4, or 8) for this UDT</td>
</tr>
<tr>
<td>locator</td>
<td>INTEGER</td>
<td>Locator key for unnamed ROW type</td>
</tr>
</tbody>
</table>

Each extended data type is characterized by a unique identifier, called an extended identifier (extended_id), a data type identifier (type), and the length and description of the data type.

For distinct types created from built-in data types, the type column codes correspond to the value of the syscolumns.coltype column (indicating the source type) as listed on page 1-18 but incremented by the hexadecimal value 0x0000800. The file $INFORMIXDIR/incl/esql/sqltypes.h contains information about sysxtdtypes.type and syscolumns.coltype codes.

An index on the extended_id column allows only unique values. An index on the locator column allows duplicate values, as does a composite indexes on the name and owner columns. A composite index on the type and source columns also allows duplicate values.

**Information Schema (IDS)**

The Information Schema consists of read-only views that provide information about all the tables, views, and columns in the current database server to which you have access. These views also provide information about SQL dialects (such as Informix, Oracle, or Sybase) and SQL standards. Note that unlike a system catalog, whose tables describes an individual database, these views describe the Dynamic Server instance, rather than a single database.
This version of the Information Schema views is an X/Open CAE standard. These standards are provided so that applications developed on other database systems can obtain Informix system catalog information without accessing the Informix system catalog tables directly.

**Important:** Because the X/Open CAE standards Information Schema views differ from ANSI-compliant Information Schema views, it is recommended that you do not install the X/Open CAE Information Schema views on ANSI-compliant databases.

The following Information Schema views are available:
- **tables**
- **columns**
- **sql_languages**
- **server_info**

Sections that follow contain information about how to generate and access Information Schema views as well as information about their structure.

**Generating the Information Schema Views**

The Information Schema views are generated automatically when you, as DBA, run the following DB–Access command:

```
dbaccess database-name $INFORMIXDIR/etc/xpg4_is.sql
```

The views display data from the system catalog tables. If tables, views, or routines exist with any of the same names as the Information Schema views, you must either rename those database objects or rename the views in the script before you can install the views. You can drop the views with the DROP VIEW statement on each view. To re-create the views, rerun the script.

**Important:** In addition to the columns specified for each Information Schema view, individual vendors might include additional columns or change the order of the columns. It is recommended that applications not use the forms SELECT * or SELECT table-name* to access an Information Schema view.

**Accessing the Information Schema Views**

All Information Schema views have the Select privilege granted to PUBLIC WITH GRANT OPTION so that all users can query the views. Because no other privileges are granted on the Information Schema views, they cannot be updated.

You can query the Information Schema views as you would query any other table or view in the database.

**Structure of the Information Schema Views**

The following Information Schema views are described in this section:
- **tables**
- **columns**
- **sql_languages**
- **server_info**
In order to accept long identifier names, most of the columns in the views are defined as VARCHAR data types with large maximum sizes.

**The tables Information Schema View**

The tables Information Schema view contains one row for each table to which you have access. It contains the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_schema</td>
<td>VARCHAR(32)</td>
<td>Name of owner of table</td>
</tr>
<tr>
<td>table_name</td>
<td>VARCHAR(128)</td>
<td>Name of table or view</td>
</tr>
<tr>
<td>table_type</td>
<td>VARCHAR(128)</td>
<td>BASE TABLE for table or VIEW for view</td>
</tr>
<tr>
<td>remarks</td>
<td>VARCHAR(255)</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>

The visible rows in the tables view depend on your privileges. For example, if you have one or more privileges on a table (such as Insert, Delete, Select, References, Alter, Index, or Update on one or more columns), or if privileges are granted to PUBLIC, you see the row that describes that table.

**The columns Information Schema View**

The columns Information Schema view contains one row for each accessible column. It contains the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>table_schema</td>
<td>VARCHAR(128)</td>
<td>Name of owner of table</td>
</tr>
<tr>
<td>table_name</td>
<td>VARCHAR(128)</td>
<td>Name of table or view</td>
</tr>
<tr>
<td>column_name</td>
<td>VARCHAR(128)</td>
<td>Name of the column in the table or view</td>
</tr>
<tr>
<td>ordinal_position</td>
<td>INTEGER</td>
<td>Position of the column within its table</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The ordinal_position value is a sequential number that starts at 1 for the first column. This is an Informix extension to XPG4.</td>
</tr>
<tr>
<td>data_type</td>
<td>VARCHAR(254)</td>
<td>Name of the data type of the column, such as CHARACTER or DECIMAL</td>
</tr>
<tr>
<td>char_max_length</td>
<td>INTEGER</td>
<td>Maximum length (in bytes) for character data types; NULL otherwise</td>
</tr>
<tr>
<td>numeric_precision</td>
<td>INTEGER</td>
<td>Uses one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Total number of digits for exact numeric data types (DECIMAL, INTEGER, MONEY, SMALLINT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Number of digits of mantissa precision (machine-dependent) for approximate data types (FLOAT, SMALLFLOAT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• NULL for all other data types.</td>
</tr>
<tr>
<td>numeric_prec_radix</td>
<td>INTEGER</td>
<td>Uses one of the following values:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 2 = Approximate data types (FLOAT and SMALLFLOAT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 10 = Exact numeric data types (DECIMAL, INTEGER, MONEY, and SMALLINT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• NULL for all other data types.</td>
</tr>
<tr>
<td>Column</td>
<td>Data Type</td>
<td>Explanation</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| numeric_scale  | INTEGER       | Number of significant digits to the right of the decimal point for DECIMAL and MONEY data types
0 for INTEGER and SMALLINT types
NULL for all other data types |
| datetime_precision | INTEGER | Number of digits in the fractional part of the seconds for DATE and DATETIME columns; NULL otherwise
This column is an Informix extension to XPG4. |
| is_nullable    | VARCHAR(3)    | Indicates whether a column allows NULL values; either YES or NO                                                                         |
| remarks        | VARCHAR(254)  | Reserved for future use                                                                                                                  |

The sql_languages Information Schema View

The sql_languages Information Schema view contains a row for each instance of conformance to standards that the current database server supports. The sql_languages view contains the following columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>VARCHAR(254)</td>
<td>Organization defining this SQL version</td>
</tr>
<tr>
<td>source_year</td>
<td>VARCHAR(254)</td>
<td>Year the source document was approved</td>
</tr>
<tr>
<td>conformance</td>
<td>VARCHAR(254)</td>
<td>Standard to which the server conforms</td>
</tr>
<tr>
<td>integrity</td>
<td>VARCHAR(254)</td>
<td>Indication of whether this is an integrity enhancement feature; either YES or NO</td>
</tr>
<tr>
<td>implementation</td>
<td>VARCHAR(254)</td>
<td>Identification of the SQL product of the vendor</td>
</tr>
<tr>
<td>binding_style</td>
<td>VARCHAR(254)</td>
<td>Direct, module, or other binding style</td>
</tr>
<tr>
<td>programming_lang</td>
<td>VARCHAR(254)</td>
<td>Host language for which binding style is adapted</td>
</tr>
</tbody>
</table>

The sql_languages view is completely visible to all users.

The server_info Information Schema View

The server_info Information Schema view describes the database server to which the application is currently connected. It contains two columns.

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>server_attribute</td>
<td>VARCHAR(254)</td>
<td>An attribute of the database server</td>
</tr>
<tr>
<td>attribute_value</td>
<td>VARCHAR(254)</td>
<td>Value of the server_attribute as it applies to the current database server</td>
</tr>
</tbody>
</table>

Each row in this view provides information about one attribute. X/Open-compliant databases must provide applications with certain required information about the database server.
The server_info view includes the following server_attribute information.

<table>
<thead>
<tr>
<th>server_attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier_length</td>
<td>Maximum number of bytes for a user-defined identifier</td>
</tr>
<tr>
<td>row_length</td>
<td>Maximum number of bytes in a row</td>
</tr>
<tr>
<td>userid_length</td>
<td>Maximum number of bytes in a user name</td>
</tr>
<tr>
<td>txn_isolation</td>
<td>Initial transaction isolation level for the database server:</td>
</tr>
<tr>
<td></td>
<td>Read Uncommitted ( = Default isolation level for databases with no transaction logging; also called Dirty Read)</td>
</tr>
<tr>
<td></td>
<td>Read Committed ( = Default isolation level for databases that are not ANSI-compliant, but that support explicit transaction logging)</td>
</tr>
<tr>
<td></td>
<td>Serializable ( = Default isolation level for ANSI-compliant databases; also called Repeatable Read)</td>
</tr>
<tr>
<td>collation_seq</td>
<td>Assumed ordering of the character set for the database server</td>
</tr>
<tr>
<td></td>
<td>The following values are possible:</td>
</tr>
<tr>
<td></td>
<td>ISO 8859-1 EBCDIC</td>
</tr>
<tr>
<td></td>
<td>The default Informix representation shows ISO 8859-1.</td>
</tr>
</tbody>
</table>

The server_info view is completely visible to all users.
Chapter 2. Data Types

In This Chapter ........................................... 2-2
Summary of Data Types .................................... 2-2
Description of Data Types ................................. 2-5
   BIGINT (XPS) ......................................... 2-5
   BLOB (IDS) ........................................... 2-5
   BOOLEAN (IDS) ...................................... 2-5
   BYTE ............................................... 2-6
   CHAR(n) ............................................. 2-7
   CHARACTER(n) ..................................... 2-8
   CHARACTER VARYING(m,r) ......................... 2-8
   CLOB (IDS) ......................................... 2-8
   DATE .............................................. 2-9
   DATETIME ......................................... 2-10
   DEC ................................................ 2-12
   DECIMAL ........................................... 2-12
   Distinct (IDS) ..................................... 2-14
   DOUBLE PRECISION ................................ 2-15
   FLOAT(n) ......................................... 2-15
   IDSSECURITYLABEL ................................. 2-15
   INT ................................................ 2-16
   INT8 .............................................. 2-16
   INTEGER .......................................... 2-16
   INTERVAL .......................................... 2-16
   LIST(e) (IDS) ..................................... 2-18
   LVARCHAR(m) (IDS) ............................... 2-19
   MONEY(p,s) ........................................ 2-20
   MULTISET(e) (IDS) ................................ 2-21
   NCHAR(n) .......................................... 2-21
   NUMERIC(p,s) ..................................... 2-22
   NVARCHAR(m,r) ................................... 2-22
   Opaque (IDS) ...................................... 2-22
   REAL .............................................. 2-22
   ROW, Named (IDS) ................................ 2-23
   ROW, Unnamed (IDS) ................................ 2-24
   SERIAL(n) .......................................... 2-25
   SERIAL8(n) ....................................... 2-26
   SET(e) (IDS) ....................................... 2-27
   SMALLFLOAT ....................................... 2-28
   SMALLINT ......................................... 2-28
   TEXT ............................................... 2-28
   VARCHAR(m,r) .................................... 2-30

Built-In Data Types ........................................ 2-31
   Large-Object Data Types ............................ 2-32
      Simple Large Objects ............................ 2-32
      Smart Large Objects (IDS) ........................ 2-32
   Time Data Types ................................ 2-33
Extended Data Types (IDS) ......................... 2-38
   Complex Data Types ............................... 2-38
      Collection Data Types .......................... 2-39
      ROW Data Types ................................ 2-40
   Distinct Data Types .............................. 2-40
   Opaque Data Types ................................ 2-40
Data Type Casting and Conversion .............. 2-41
   Using Built-in Casts .............................. 2-41
      Converting from Number to Number ........... 2-42

© Copyright IBM Corp. 1996, 2007
In This Chapter

Every column in a table in a database is assigned a data type. The data type precisely defines the kinds of values that you can store in that column.

This chapter describes built-in and extended data types, casting between two data types, and operator precedence.

Summary of Data Types

Figure 2-1 shows the logical categories of data types that Informix database servers support. Shaded categories are for Dynamic Server only.

![Data types diagram]

Figure 2-1. Overview of Supported Data Types

This diagram is simplified; some built-in types are implemented as opaque types, and are only supported on Dynamic Server. That is, opaque and built-in are not disjunct categories, though most built-in data types are not opaque.

Built-in data types (which are system-defined) and extended data types (which you can define) share the following characteristics. You can:

- Use them to create columns within database tables.
- Declare them as arguments and as returned types of routines.
- Use them as base types from which to create DISTINCT data types.
- Cast them to other data types.
- Declare and access host variables of these types in SPL and ESQL/C.

For exceptions, see the description of each data type. For an overview, see "Built-In Data Types" on page 2-31 and "Extended Data Types (IDS)" on page 2-38.
You assign data types to columns with the CREATE TABLE statement and change them with the ALTER TABLE statement. When you change an existing column data type, all data is converted to the new data type, if possible.

For information on the ALTER TABLE and CREATE TABLE statements, on SQL statements that create specific data types, that create and drop casts, and on other data type topics, refer to the IBM Informix Guide to SQL: Syntax.

For information about how to create and use complex data types of Dynamic Server, see the IBM Informix Database Design and Implementation Guide. For information about how to create user-defined data types, see IBM Informix User-Defined Routines and Data Types Developer's Guide.

All Informix database servers support the data types that Table 2-1 lists. This chapter describes each of these built-in data types.

Table 2-1. Data Types That All Informix Database Servers Support

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Explanation</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>Is a synonym (in XPS only) for INT8</td>
<td>2-16</td>
</tr>
<tr>
<td>BYTE</td>
<td>Stores any kind of binary data, up to 2^31 bytes in length</td>
<td>2-4</td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>Stores character strings; collation is in code-set order</td>
<td>2-7</td>
</tr>
<tr>
<td>CHARACTER(n)</td>
<td>Is a synonym for CHAR</td>
<td>2-8</td>
</tr>
<tr>
<td>CHARACTER VARYING(m, r)</td>
<td>Stores character strings of varying length (ANSI compliant); collation is in code-set order</td>
<td>2-8</td>
</tr>
<tr>
<td>DATE</td>
<td>Stores calendar dates</td>
<td>2-9</td>
</tr>
<tr>
<td>DATETIME</td>
<td>Stores calendar date combined with time of day</td>
<td>2-10</td>
</tr>
<tr>
<td>DEC</td>
<td>Is a synonym for DECIMAL</td>
<td>2-12</td>
</tr>
<tr>
<td>DECIMAL(p)</td>
<td>Stores floating-point numbers with definable precision; if database is ANSI-compliant, the scale is zero.</td>
<td>2-12</td>
</tr>
<tr>
<td>DECIMAL(p, s)</td>
<td>Stores fixed-point numbers of defined scale and precision</td>
<td>2-13</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>Synonym for FLOAT</td>
<td>2-15</td>
</tr>
<tr>
<td>FLOAT(n)</td>
<td>Stores double-precision floating-point numbers corresponding to the double data type in C</td>
<td>2-15</td>
</tr>
<tr>
<td>IDSSecurityLabel</td>
<td>Defines the row security label column.</td>
<td>2-15</td>
</tr>
<tr>
<td>INT</td>
<td>Is a synonym for INTEGER</td>
<td>2-16</td>
</tr>
<tr>
<td>INT8</td>
<td>Stores 8-byte integer values in range -2^63 -1 to 2^63 -1</td>
<td>2-16</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Stores whole numbers in a range from -2,147,483,647 to +2,147,483,647</td>
<td>2-16</td>
</tr>
<tr>
<td>INTERVAL (Year</td>
<td>Stores a span of time (or level of effort) in units of years and months.</td>
<td>2-16</td>
</tr>
<tr>
<td>(Month)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTERVAL (Day</td>
<td>Stores a span of time in a contiguous set of units of days, hours, minutes, seconds, and fractions of a second.</td>
<td>2-16</td>
</tr>
<tr>
<td>Fraction)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MONEY(p,s)</td>
<td>Stores currency amounts</td>
<td>2-20</td>
</tr>
<tr>
<td>NCHAR(n)</td>
<td>Same as CHAR, but can support localized collation</td>
<td>2-21</td>
</tr>
<tr>
<td>NUMERIC(p,s)</td>
<td>Synonym for DECIMAL(p,s)</td>
<td>2-22</td>
</tr>
<tr>
<td>NVARCHAR(m,r)</td>
<td>Same as VARCHAR, but can support localized collation</td>
<td>2-22</td>
</tr>
<tr>
<td>REAL</td>
<td>Is a synonym for SMALLFLOAT</td>
<td>2-22</td>
</tr>
<tr>
<td>SERIAL</td>
<td>Stores sequential integers (&gt; 0) in positive range of INT</td>
<td>2-25</td>
</tr>
</tbody>
</table>
Table 2-1. Data Types That All Informix Database Servers Support (continued)

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Explanation</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SERIAL8</td>
<td>Stores sequential integers ( &gt; 0) in positive range of INT8</td>
<td>2-26</td>
</tr>
<tr>
<td>SMALLFLOAT</td>
<td>Stores single-precision floating-point numbers corresponding to the float data type of the C language</td>
<td>2-28</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>Stores whole numbers from -32,767 to +32,767</td>
<td>2-28</td>
</tr>
<tr>
<td>TEXT</td>
<td>Stores any kind of text data, up to 2^31 bytes in length</td>
<td>2-28</td>
</tr>
<tr>
<td>VARCHAR(m,r)</td>
<td>Stores character strings of varying length (up to 255 bytes); collation is in code-set order.</td>
<td>2-30</td>
</tr>
</tbody>
</table>

For the character data types (CHAR, CHAR VARYING, LVARCHAR, NCHAR, NVARCHAR, and VARCHAR), a data string can include letters, digits, punctuation, whitespace, diacritical marks, ligatures, and other printable symbols from the code set of the database locale. (For some East Asian locales, multibyte characters are supported within data strings.)

Dynamic Server also supports additional data types that Table 2-2 lists.

Table 2-2. Additional Data Types That Dynamic Server Supports

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Explanation</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB</td>
<td>Stores binary data in random-access chunks</td>
<td>2-5</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>Stores Boolean values true and false</td>
<td>2-5</td>
</tr>
<tr>
<td>CLOB</td>
<td>Stores text data in random-access chunks</td>
<td>2-8</td>
</tr>
<tr>
<td>Distinct</td>
<td>Stores data in a user-defined type that has the same format as a source type on which it is based, but its casts and functions can differ from those on the source type</td>
<td>2-14</td>
</tr>
<tr>
<td>LIST(e)</td>
<td>Stores a sequentially ordered collection of elements, all of the same data type, e allows duplicate values</td>
<td>2-18</td>
</tr>
<tr>
<td>LVARCHAR(m)</td>
<td>Stores variable-length strings of up to 32,739 bytes</td>
<td>2-19</td>
</tr>
<tr>
<td>MULTISET(e)</td>
<td>Stores a non-ordered collection of values, with elements all of the same data type, e allows duplicate values.</td>
<td>2-21</td>
</tr>
<tr>
<td>Opaque</td>
<td>Stores a user-defined data type whose internal structure is inaccessible to the database server</td>
<td>2-22</td>
</tr>
<tr>
<td>ROW, Named</td>
<td>Stores a named ROW type</td>
<td>2-23</td>
</tr>
<tr>
<td>ROW, Unnamed</td>
<td>Stores an unnamed ROW type</td>
<td>2-24</td>
</tr>
<tr>
<td>SET(e)</td>
<td>Stores a non-ordered collection of elements, all of the same data type, e does not allow duplicate values</td>
<td>2-27</td>
</tr>
</tbody>
</table>

Distributed DML operations and function calls that access databases of other database servers cannot return these extended data types of Dynamic Server, which are individually described in this chapter. (Cross-database operations on other databases of the same Dynamic Server instance, however, can access BOOLEAN, BLOB, CLOB, and LVARCHAR data types, which are implemented as built-in opaque types. Such operations can also access DISTINCT types whose base types are built-in types, and user-defined types (UDTs), if the UDTs and DISTINCT types are explicitly cast to built-in types, and if all of the UDTs, casts, and DISTINCT types are defined in all the participating databases.)
For information about Informix internal data types that SQL statements support (such as IMPEX, IMPEXBIN, and SENDRECV), see IBM Informix User-Defined Routines and Data Types Developer’s Guide.

Description of Data Types

This section describes the data types that Informix database servers support.

**BIGINT (XPS)**

The BIGINT data type of Extended Parallel Server is a synonym for INT8. (See the description of INT8 on page 2-16)

**BLOB (IDS)**

The BLOB data type stores any kind of binary data in random-access chunks, called sbspaces. Binary data typically consists of saved spreadsheets, program-load modules, digitized voice patterns, and so on. The database server performs no interpretation of the contents of a BLOB column. A BLOB column can be up to 4 terabytes (4*2^30 bytes) in length, though your system resources might impose a lower practical limit.

The term *smart large object* refers to BLOB and CLOB data types. Use CLOB data types (see page 2-8) for random access to text data. For general information about BLOB and CLOB data types, see “Smart Large Objects (IDS)” on page 2-32.

You can use these SQL functions to perform operations on a BLOB column:

- **FILETOBLOB** copies a file into a BLOB column.
- **LOTOFILE** copies a BLOB (or CLOB) value into an operating-system file.
- **LOCOPY** copies an existing smart large object to a new smart large object.

For more information on these SQL functions, see the IBM Informix Guide to SQL: Syntax.

Within SQL, you are limited to the equality ( = ) comparison operation and the encryption and decryption functions for BLOB data. (The encryption and decryption functions are described in the IBM Informix Guide to SQL: Syntax.) To perform additional operations, you must use one of the application programming interfaces (APIs) from within your client application.

You can insert data into BLOB columns in the following ways:

- With the dbload or onload utilities
- With the LOAD statement (DB–Access)
- With the FILETOBLOB function
- From BLOB (ifix_lo_t) host variables (IBM Informix ESQL/C)

If you select a BLOB column using DB–Access, only the string <SBlob value> is returned; no actual value is displayed.

**BOOLEAN (IDS)**

The BOOLEAN data type stores TRUE or FALSE data values as a single byte. This table shows internal and literal representations of the BOOLEAN data type.
You can compare two BOOLEAN values to test for equality or inequality. You can also compare a BOOLEAN value to the Boolean literals ‘t’ and ‘f’. BOOLEAN values are case insensitive; ‘t’ is equivalent to ‘T’ and ‘f’ to ‘F’.

You can use a BOOLEAN column to store what a Boolean expression returns. In the following example, the value of boolean_column is ‘t’ if column1 is less than column2, ‘f’ if column1 is greater than or equal to column2, and NULL if the value of either column1 or column2 is unknown:
UPDATE my_table SET boolean_column = lessthan(column1, column2)

**BYTE**

The BYTE data type stores any kind of binary data in an undifferentiated byte stream. Binary data typically consists of digitized information, such as spreadsheets, program load modules, digitized voice patterns, and so on. The term simple large object refers to BYTE and text data types. No more than 195 columns of the same table can be declared as BYTE and text data types.

The BYTE data type has no maximum size. A BYTE column has a theoretical limit of \(2^{64}\) bytes and a practical limit that your disk capacity determines.

You can store, retrieve, update, or delete the contents of a BYTE column. You cannot, however, use BYTE operands in arithmetic or string operations, nor assign literals to BYTE columns with the SET clause of the UPDATE statement. You also cannot use BYTE items in any of the following ways:
- With aggregate functions
- With the IN clause
- With the MATCHES or LIKE clauses
- With the GROUP BY clause
- With the ORDER BY clause

BYTE operands are valid in Boolean expressions only when you are testing for NULL values with the IS NULL or IS NOT NULL operators.

You can insert data into BYTE columns in the following ways:
- With the dbload or onload utilities
- With the LOAD statement (DB–Access)
- From BYTE host variables (IBM Informix ESQL/C)

You cannot use a quoted text string, number, or any other actual value to insert or update BYTE columns.

When you select a BYTE column, you can choose to receive all or part of it. To retrieve it all, use the regular syntax for selecting a column. You can also select any part of a BYTE column by using subscripts, as the next example, which reads the first 75 bytes of the cat_picture column associated with the catalog number 10001:
SELECT cat_picture [1,75] FROM catalog WHERE catalog_num = 10001
A built-in cast converts BYTE values to BLOB values. For more information, see the IBM Informix Database Design and Implementation Guide.

If you select a BYTE column using the DB–Access Interactive Schema Editor, only the string “<BYTE value>” is returned; no data value is displayed.

Important: If you try to return a BYTE column from a subquery, an error results, even if the column is not used in a Boolean expression nor with an aggregate.

**CHAR(n)**

The CHAR data type stores any string of letters, numbers, and symbols. It can store single-byte and multibyte characters, based on the database locale. (For more information on East Asian locales that support multibyte code sets, see "Multibyte Characters with VARCHAR” on page 2:31)

A CHAR(n) column has a length of n bytes, where 1 ≤ n ≤ 32,767. If you do not specify n, CHAR(1) is the default length. Character columns typically store alphanumeric strings, such as names, addresses, phone numbers, and so on. When a value is retrieved or stored as CHAR(n), exactly n bytes of data are transferred. If the string is shorter than n bytes, the string is extended with blank spaces up to the declared length. If the data value is longer than n bytes, a data string of length n that has been truncated from the right is inserted or retrieved, without the database server raising an exception.

This does not create partial characters in multibyte locales. In right-to-left locales, such as Arabic, Hebrew, or Farsi, the truncation is from the left.

**Treating CHAR Values as Numeric Values**

If you plan to perform calculations on numbers stored in a column, you should assign a number data type to that column. Although you can store numbers in CHAR columns, you might not be able to use them in some arithmetic operations. For example, if you insert a sum into a CHAR column, you might experience overflow problems if the CHAR column is too small to hold the value. In this case, the insert fails. Numbers that have leading zeros (such as some zip codes) have the zeros stripped if they are stored as number types INTEGER or SMALLINT. Instead, store these numbers in CHAR columns.

**Sorting and Relational Comparisons**

In general, the collating order for sorting CHAR values is the order of characters in the code set. (An exception is the MATCHES operator with ranges; see "Collating VARCHAR Values” on page 2:31) For more information about collation order, see the IBM Informix GLS User’s Guide.

For multibyte locales, the database supports any multibyte characters in the code set. When storing multibyte characters in a CHAR data type, make sure to calculate the number of bytes needed. For more information on multibyte characters and locales, see the IBM Informix GLS User’s Guide.

CHAR values are compared to other CHAR values by padding the shorter value on the right with blank spaces until the values have equal length, and then comparing the two values, using the code-set order for collation.

**Nonprintable Characters with CHAR**

A CHAR value can include tab, newline, whitespace, and nonprintable characters. You must, however, use an application to insert nonprintable characters into host
variables and the host variables into your database. After passing nonprintable
characters to the database server, you can store or retrieve them. After you select
nonprintable characters, fetch them into host variables and display them with your
own display mechanism.

An important exception is the first value in the ASCII code set is used as the
end-of-data terminator symbol in columns of the CHAR data type. For this reason,
any subsequent characters in the same string cannot be retrieved from a CHAR
column, because the database server reads only the characters (if any) that precede
this null terminator. For example, you cannot use the following 7-byte string as a
CHAR data type value with a length of 7 bytes:

```
abc\0def
```

If you try to display nonprintable characters with DB–Access your screen returns
inconsistent results. (Which characters are nonprintable is locale-dependent. For
more information see the discussion of code-set conversion between the client and
the database server in the IBM Informix GLS User’s Guide.)

**CHARACTER(n)**

The CHARACTER data type is a synonym for CHAR.

**CHARACTER VARYING(m,r)**

The CHARACTER VARYING data type stores a string of letters, digits, and
symbols of varying length, where \( m \) is the maximum size of the column (in bytes)
and \( r \) is the minimum number of bytes reserved for that column. The
CHARACTER VARYING data type complies with ANSI/ISO standard for SQL; the
non-ANSI VARCHAR data type supports the same functionality. For more
information, see the description of the VARCHAR type in [VARCHAR(m,r)” on
page 2-30](#).

**CLOB (IDS)**

The CLOB data type stores any kind of text data in random-access chunks, called
sbspaces. Text data can include text-formating information, as long as this
information is also textual, such as PostScript, Hypertext Markup Language
(HTML), Standard Graphic Markup Language (SGML), or Extensible Markup
Language (XML) data.

The term *smart large object* refers to CLOB and BLOB data types. The CLOB data
type supports special operations for character strings that are inappropriate for
BLOB values. A CLOB value can be up to 4 terabytes \( (4\times2^{40} \text{ bytes}) \) in length.

Use the BLOB data type (see [“BLOB (IDS)” on page 2-5](#)) for random access
to binary data. For general information about the CLOB and BLOB data types, see
[“Smart Large Objects (IDS)” on page 2-32](#)

The following SQL functions can perform operations on a CLOB column:

- **FILETOCLOB** copies a file into a CLOB column.
- **LOTOFILE** copies a CLOB (or BLOB) value into a file.
- **LOCOPY** copies a CLOB (or BLOB) value to a new smart large object.
- **ENCRYPT_DES** or **ENCRYPT_TDES** creates an encrypted BLOB value from a
plain-text CLOB argument.
• **DECRIPT_BINAR** or **DECRIPT_CHAR** returns an unencrypted BLOB value from an encrypted BLOB argument (that **ENCRIPT_DES** or **ENCRIPT_TDES** created from a plain-text CLOB value).

For more information on these SQL functions, see the *IBM Informix Guide to SQL: Syntax*.

No casts exist for CLOB data. Therefore, the database server cannot convert data of the CLOB type to any other data type, except by using these encryption and decryption functions to return a BLOB. Within SQL, you are limited to the equality (=) comparison operation for CLOB data. To perform additional operations, you must use one of the application programming interfaces from within your client application.

**Multibyte Characters with CLOB**

You can insert data into CLOB columns in the following ways:

• With the **dbload** or **onload** utilities
• With the LOAD statement (DB–Access)
• From CLOB (ifx_lo_t) host variables (ESQL/C).

For examples of CLOB types, see the *IBM Informix Guide to SQL: Tutorial* and the *IBM Informix Database Design and Implementation Guide*.

With GLS, the following rules apply:

• Multibyte CLOB characters must be defined in the database locale.
• The CLOB data type is collated in code-set order.
• The database server handles code-set conversions for CLOB data.

For more information on database locales, collation order, and code-set conversion, see the *IBM Informix GLS User’s Guide*.

**DATE**

The DATE data type stores the calendar date. DATE data types require four bytes. A calendar date is stored internally as an integer value equal to the number of days since December 31, 1899.

Because DATE values are stored as integers, you can use them in arithmetic expressions. For example, you can subtract a DATE value from another DATE value. The result, a positive or negative INTEGER value, indicates the number of days that elapsed between the two dates. (You can use a UNITS DAY expression to convert the result to an INTERVAL DAY TO DAY data type.)

The following example shows the default display format of a DATE column:

`mm/dd/yyyy`

In this example, `mm` is the month (1-12), `dd` is the day of the month (1-31), and `yyyy` is the year (0001-9999). You can specify a different order of time units and a different time-unit separator than / (or no separator) by setting the **DBDATE** environment variable. For more information, see "**DBDATE**" on page 3-20.

In non-default locales, you can display dates in culture-specific formats. The locale and the **GL_DATE** and **DBDATE** environment variables (as described in the next chapter) affect the display formatting of DATE values. They do not, however, affect
DATETIME

The DATETIME data type stores an instant in time expressed as a calendar date and time of day. You choose how precisely a DATETIME value is stored; its precision can range from a year to a fraction of a second.

DATETIME stores a data value as a contiguous series of fields that represents each time unit (year, month, day, and so forth) in the data type declaration.

Field qualifiers to specify a DATETIME data type have this format:

DATETIME largest_qualifier TO smallest_qualifier

This resembles an INTERVAL field qualifier (see \[^{\text{INTERVAL}}\text{ on page 2-16}\] ), but DATETIME represents a point in time, rather than (like INTERVAL) a span of time. These differences exist between DATETIME and INTERVAL qualifiers:

- The DATETIME keyword replaces the INTERVAL keyword.
- DATETIME field qualifiers cannot specify a non-default precision for the largest_qualifier time unit.
- A DATETIME value that includes YEAR and/or MONTH time units can also include smaller time units, whereas an INTERVAL data type that stores days (or smaller time units) cannot store months or years.

The largest_qualifier and smallest_qualifier of a DATETIME data type can be any of the fields that \[^{\text{Table 2-3}}\] lists, provided that smallest_qualifier does not specify a larger time unit than largest_qualifier. (The largest and smallest time units can be the same; for example, DATETIME YEAR TO YEAR.)

\[\begin{array}{|c|c|}
\hline
\text{Qualifier Field} & \text{Valid Entries} \\
\hline
\text{YEAR} & \text{A year numbered from 1 to 9,999 (A.D.)} \\
\hline
\text{MONTH} & \text{A month numbered from 1 to 12} \\
\hline
\text{DAY} & \text{A day numbered from 1 to 31, as appropriate to the month} \\
\hline
\text{HOUR} & \text{An hour numbered from 0 (midnight) to 23} \\
\hline
\text{MINUTE} & \text{A minute numbered from 0 to 59} \\
\hline
\text{SECOND} & \text{A second numbered from 0 to 59} \\
\hline
\text{FRACTION} & \text{A decimal fraction-of-a-second with up to 5 digits of scale. The default scale is 3 digits (a thousandth of a second). For smallest_qualifier to specify another scale, write FRACTION(n), where n is the desired number of digits from 1 to 5.} \\
\hline
\end{array}\]

The declaration of a DATETIME column need not include the full YEAR to FRACTION range of time units. It can include any contiguous subset of these time units, or even only a single time unit.

For example, you can enter a MONTH TO HOUR value in a column declared as YEAR TO MINUTE, as long as each entered value contains information for a contiguous series of time units. You cannot, however, enter a value for only the MONTH and HOUR; the entry must also include a value for DAY.
If you use the DB–Access TABLE menu, and you do not specify the DATETIME
qualifiers, a default DATETIME qualifier, YEAR TO YEAR, is assigned.

A valid DATETIME literal must include the DATETIME keyword, the values to be
entered, and the field qualifiers. You must include these qualifiers because, as
noted earlier, the value that you enter can contain fewer fields than were declared
for that column. Acceptable qualifiers for the first and last fields are identical to
the list of valid DATETIME fields that Table 2-3 on page 2-10 lists.

Write values for the field qualifiers as integers and separate them with delimiters.
Table 2-4 lists the delimiters that are used with DATETIME values in the default
U.S. English locale. (These are a superset of the delimiters that are used in
INTERVAL values; see Table 2-6 on page 2-18.)

Table 2-4. Delimiters Used with DATETIME

<table>
<thead>
<tr>
<th>Delimiter</th>
<th>Placement in DATETIME Literal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyphen (-)</td>
<td>Between the YEAR, MONTH, and DAY time-unit values</td>
</tr>
<tr>
<td>Blank space (</td>
<td>Between the DAY and HOUR time-unit values</td>
</tr>
<tr>
<td>Colon (:)</td>
<td>Between the HOUR, MINUTE, and SECOND time-unit values</td>
</tr>
<tr>
<td>Decimal point (.)</td>
<td>Between the SECOND and FRACTION time-unit values</td>
</tr>
</tbody>
</table>

Figure 2-2 shows a DATETIME YEAR TO FRACTION(3) value with delimiters.

![Figure 2-2](image)

Figure 2-2. Example DATETIME Value with Delimiters

When you enter a value with fewer time-unit fields than in the column, the value
that you enter is expanded automatically to fill all the declared time-unit fields. If
you leave out any more significant fields, that is, time units larger than any that
you include, those fields are filled automatically with the current values for those
time units from the system clock calendar. If you leave out any less-significant
fields, those fields are filled with zeros (or with 1 for MONTH and DAY) in your
entry.

You can also enter DATETIME values as character strings. The character string
must include information for each field defined in the DATETIME column. The
INSERT statement in the following example shows a DATETIME value entered as
a character string:

```sql
INSERT INTO cust_calls (customer_num, call_dtime, user_id,
call_code, call_descr)
VALUES (101, '2001-01-14 08:45', 'maryj', '0','
'Order late - placed 6/1/00')
```

If `call_dtime` is declared as DATETIME YEAR TO MINUTE, the character string
must include values for the `year, month, day, hour, and minute` fields.

If the character string does not contain information for all the declared fields (or if
it adds additional fields), then the database server returns an error.
All fields of a DATETIME column are two-digit numbers except for the year and fraction fields. The year field is stored as four digits. When you enter a two-digit value in the year field, how the abbreviated year is expanded to four digits depends on the setting of the DBCENTURY environment variable.

For example, if you enter 02 as the year value, whether the year is interpreted as 1902, 2002, or 2102 depends on the setting of DBCENTURY and on the value of the system clock calendar at execution time. If you do not set DBCENTURY, the leading digits of the current year are appended by default. For information about setting DBCENTURY, see “DBCENTURY” on page 3-18.

The fraction field requires \( n \) digits where \( 1 \leq n \leq 5 \), rounded up to an even number. You can use the following formula (rounded up to a whole number of bytes) to calculate the number of bytes that a DATETIME value requires:

\[
\text{(total number of digits for all fields)} / 2 + 1
\]

For example, a YEAR TO DAY qualifier requires a total of eight digits (four for year, two for month, and two for day). According to the formula, this data value requires 5, or \((8/2) + 1\), bytes of storage.

For information on how to use DATETIME values in arithmetic and relational expressions, see “Manipulating DATE with DATETIME and INTERVAL Values” on page 2-35. For more information on the DATETIME data type see the IBM Informix Guide to SQL: Syntax.

If you specify a locale other than U.S. English, the locale defines the culture-specific display formats for DATETIME values. To change the default display format, change the setting of the GL_DATETIME environment variable.

With an ESQL API, the DBTIME environment variable also affects DATETIME formatting. Non-default locales and settings of the GL_DATE and DBDATE environment variables also affect the display of datetime data. They do not, however, affect the internal storage format of a DATETIME column.

The USEOSTIME configuration parameter can affect the subsecond granularity when the database server obtains the current time from the operating system in SQL statements; for details, see the IBM Informix Administrator’s Reference.

For more information on DBTIME, see “DBTIME” on page 3-31. For more information on DBCENTURY, see “DBCENTURY” on page 3-18. For more information on locales and GLS environment variables that can specify end-user DATETIME formats, see the IBM Informix GLS User’s Guide.

DEC

The DEC data type is a synonym for DECIMAL.

DECIMAL

The DECIMAL data type can take two forms: DECIMAL\((p)\) floating point and DECIMAL\((p,s)\) fixed point. In an ANSI-compliant database, however, all DECIMAL numbers are fixed point. By default, literal numbers that include a decimal (.) point are interpreted by the database server as DECIMAL values.
**DECIMAL(p) Floating Point**

The DECIMAL data type stores decimal floating-point numbers up to a maximum of 32 significant digits, where \( p \) is the total number of significant digits (the *precision*).

Specifying precision is optional. If you specify no precision \( (p) \), DECIMAL is treated as DECIMAL(16), a floating-point decimal with a precision of 16 places. DECIMAL\( (p) \) has an absolute exponent range between \( 10^{-130} \) and \( 10^{124} \).

If you declare a DECIMAL\( (p) \) column in an ANSI-compliant database, the scale defaults to DECIMAL\( (p, 0) \), meaning that only integer values can be stored in this data type.

In a database that is not ANSI-compliant, a DECIMAL\( (p) \) is a floating-point data type of a scale large enough to store the exponential notation for a value.

For example, the following calculation shows how many bytes of storage a DECIMAL\( (5) \) column requires in the default locale (where the decimal point occupies a single byte):

1 byte for the sign of the data value
1 byte for the first digit
1 byte for the decimal point
4 bytes for the rest of the digits in the declared precision of \( (5) \) - 1
1 byte for the 'e' symbol
1 byte for the sign of the exponent
3 bytes for the exponent
------------------------------------------
12 bytes (Total)

Thus, "12345" in a DECIMAL\( (5) \) column is displayed as "12345.00000" (that is, with a scale of 6) in a database that is not ANSI-compliant.

**DECIMAL \( (p,s) \) Fixed Point**

In fixed-point numbers, DECIMAL\( (p,s) \), the decimal point is fixed at a specific place, regardless of the value of the number. When you specify a column of this type, you declare its precision \( (p) \) as the total number of digits that it can store, from 1 to 32. You declare its *scale* \( (s) \) as the total number of digits in the fractional part (that is, to the right of the decimal point).

All numbers with an absolute value less than \( 0.5 \times 10^s \) have the value zero. The largest absolute value of a DECIMAL\( (p,s) \) data type that you can store without an overflow error is \( 10^p \times -10^s \). A DECIMAL column typically stores numbers with fractional parts that must be stored and displayed exactly (for example, rates or percentages). In an ANSI-compliant database, all DECIMAL numbers must have absolute values in the range \( 10^{-32} \) to \( 10^{31} \).

**DECIMAL Storage**

The database server uses one byte of disk storage to store two digits of a decimal number, plus an additional byte to store the exponent and sign, with the first byte representing a sign bit and a 7-bit exponent in excess-65 format. The rest of the bytes express the mantissa as base-100 digits. The significant digits to the left of the decimal and the significant digits to the right of the decimal are stored in separate groups of bytes. At the maximum precision specification, DECIMAL\( (32,s) \) data types can store \( s-1 \) decimal digits to the right of the decimal point, if \( s \) is an odd number.
How the database server stores decimal numbers is illustrated in the following example. If you specify DECIMAL(6,3), the data type consists of three significant digits in the integral part and three significant digits in the fractional part (for instance, 123.456). The three digits to the left of the decimal are stored on 2 bytes (where one of the bytes only holds a single digit) and the three digits to the right of the decimal are stored on another 2 bytes, as Figure 2.3 illustrates.

(The exponent byte is not shown.) With the additional byte required for the exponent and sign, DECIMAL(6,3) requires a total of 5 bytes of storage.

![Figure 2-3. Schematic That Illustrates the Storage of Digits in a Decimal (p,s) Value](image)

You can use the following formulas (rounded down to a whole number of bytes) to calculate the byte storage (N) for a DECIMAL(p,s) data type (where N includes the byte that is required to store the exponent and the sign):

- If the scale is odd: \( N = (\text{precision} + 4) / 2 \)
- If the scale is even: \( N = (\text{precision} + 3) / 2 \)

For example, the data type DECIMAL(5,3) requires 4 bytes of storage (9/2 rounded down equals 4).

There is one caveat to these formulas. The maximum number of bytes the database server uses to store a decimal value is 17. One byte is used to store the exponent and sign, leaving 16 bytes to store up to 32 digits of precision. If you specify a precision of 32 and an odd scale, however, you lose 1 digit of precision. Consider, for example, the data type DECIMAL(32,31). This decimal is defined as 1 digit to the left of the decimal and 31 digits to the right. The 1 digit to the left of the decimal requires 1 byte of storage. This leaves only 15 bytes of storage for the digits to the right of the decimal. The 15 bytes can accommodate only 30 digits, so 1 digit of precision is lost.

**Distinct (IDS)**

A **distinct** type is a data type that is derived from one of the following source types (called the base type):

- A built-in type
- An existing distinct type
- An existing named ROW type
- An existing opaque type

A distinct type inherits from its source type the length and alignment on the disk. A distinct type thus makes efficient use of the preexisting functionality of the database server.

When you create a distinct data type, the database server automatically creates two explicit casts: one cast from the distinct type to its source type and one cast from the source type to the distinct type. A distinct type based on a built-in source type
does not inherit the built-in casts that are provided for the built-in type. A distinct type does inherit, however, any user-defined casts that have been defined on the source type.

A distinct type cannot be compared directly to its source type. To compare the two types, you must first explicitly cast one type to the other.

You must define a distinct type in the database. Definitions of distinct types are stored in the syxstdtypes system catalog table. The following SQL statements maintain the definitions of distinct types in the database:

- The CREATE DISTINCT TYPE statement adds a distinct type to the database.
- The DROP TYPE statement removes a previously defined distinct type from the database.

For more information about the SQL statements mentioned above, see the *IBM Informix Guide to SQL: Syntax*. For information about casting distinct data types, see “Casts for Distinct Types” on page 2-44. For examples that show how to create and register cast functions for a distinct type, see the *IBM Informix Database Design and Implementation Guide*.

**DOUBLE PRECISION**

The DOUBLE PRECISION keywords are a synonym for the FLOAT keyword.

**FLOAT(n)**

The FLOAT data type stores double-precision floating-point numbers with up to 17 significant digits. FLOAT corresponds to IEEE 4-byte floating-point, and to the `double` data type in C. The range of values for the FLOAT data type is the same as the range of the C `double` data type on your computer.

You can use `n` to specify the precision of a FLOAT data type, but SQL ignores the precision. The value `n` must be a whole number between 1 and 14.

A column with the FLOAT data type typically stores scientific numbers that can be calculated only approximately. Because floating-point numbers retain only their most significant digits, the number that you enter in this type of column and the number the database server displays can differ slightly.

The difference between the two values depends on how your computer stores floating-point numbers internally. For example, you might enter a value of 1.1000001 into a FLOAT field and, after processing the SQL statement, the database server might display this value as 1.1. This situation occurs when a value has more digits than the floating-point number can store. In this case, the value is stored in its approximate form with the least significant digits treated as zeros.

FLOAT data types usually require 8 bytes of storage per value. Conversion of a FLOAT value to a DECIMAL value results in 17 digits of precision.

**IDSSECURITYLABEL**

The IDSSECURITYLABEL type stores a security label in a table that is protected by a security policy. Only a user who holds the DBSECADM role can create, alter, or drop a column of this data type. IDSSECURITYLABEL is a built-in DISTINCT OF VARCHAR(128) data type. A table that has a security policy can have only one IDSSECURITYLABEL column. A table with no security policy can have none. You cannot encrypt the security label in a column of type IDSSECURITY label.
**INT**

The INT data type is a synonym for INTEGER.

**INT8**

The INT8 data type stores whole numbers that can range in value from -9,223,372,036,854,775,807 to 9,223,372,036,854,775,807 [or \((-2^{63}) \text{ to } 2^{63}\)] for 18 or 19 digits of precision. The number -9,223,372,036,854,775,808 is a reserved value that cannot be used. The INT8 data type is typically used to store large counts, quantities, and so on.

Dynamic Server stores INT8 data in internal format that can require up to 10 bytes of storage. Extended Parallel Server stores INT8 values as 8 bytes.

Arithmetic operations and sort comparisons are performed more efficiently on integer data than on floating-point or fixed-point decimal data, but INT8 cannot store data with absolute values beyond \(2^{63}-1\). If a value exceeds the numeric range of INT8, the database server does not store the value.

**INTEGER**

The INTEGER data type stores whole numbers that range from -2,147,483,647 to 2,147,483,647 for 9 or 10 digits of precision. The number 2,147,483,648 is a reserved value and cannot be used. The INTEGER value is stored as a signed binary integer and is typically used to store counts, quantities, and so on.

Arithmetic operations and sort comparisons are performed more efficiently on integer data than on float or decimal data. INTEGER columns, however, cannot store absolute values beyond \(2^{31}-1\). If a data value lies outside the numeric range of INTEGER, the database server does not store the value.

INTEGER data types require 4 bytes of storage per value.

**INTERVAL**

The INTERVAL data type stores a value that represents a span of time. INTERVAL types are divided into two classes: *year-month intervals* and *day-time intervals*. A year-month interval can represent a span of years and months, and a day-time interval can represent a span of days, hours, minutes, seconds, and fractions of a second.

An INTERVAL value is always composed of one value or a series of values that represents time units. Within a data-definition statement such as CREATE TABLE or ALTER TABLE that defines the precision of an INTERVAL data type, the qualifiers must have the following format:

INTERVAL *largest_qualifier*(n) TO *smallest_qualifier*

Here the *largest_qualifier* and *smallest_qualifier* keywords are taken from one of the two INTERVAL classes, as shown in Table 2-5 on page 2-17.

If SECOND (or a larger time unit) is the *largest_qualifier*, the declaration of an INTERVAL data type can optionally specify n, the precision of the largest time unit (for n ranging from 1 to 9); this is not a feature of DATETIME data types.

If *smallest_qualifier* is FRACTION, you can also specify a scale in the range from 1 to 5. For FRACTION TO FRACTION qualifiers, the upper limit of n is 5, rather than 9. There are two incommensurable classes of INTERVAL data types:
• Those with a smallest _qualifier_ larger than DAY
• Those with a largest _qualifier_ smaller than MONTH

<table>
<thead>
<tr>
<th>Interval Class</th>
<th>Time Units</th>
<th>Valid Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEAR-MONTH</td>
<td>YEAR</td>
<td>A number of years</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>MONTH</td>
<td>A number of months</td>
</tr>
<tr>
<td>DAY-TIME INTERVAL</td>
<td>DAY</td>
<td>A number of days</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>HOUR</td>
<td>A number of hours</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>MINUTE</td>
<td>A number of minutes</td>
</tr>
<tr>
<td>INTERVAl</td>
<td>SECOND</td>
<td>A number of seconds</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>FRACTION</td>
<td>A decimal fraction of a second, with up to 5 digits. The default scale is 3 digits (thousandth of a second). To specify a non-default scale, write FRACTION(n), where 1 ≤ n ≤ 5.</td>
</tr>
</tbody>
</table>

As with DATETIME data types, you can define an INTERVAL to include only the subset of time units that you need. But because the construct of “month” (as used in calendar dates) is not a time unit that has a fixed number of days, a single INTERVAL value cannot combine months and days; arithmetic that involves operands of the two different INTERVAL classes is not supported.

A value entered into an INTERVAL column need not include the full range of time units that were specified in the data-type declaration of the column. For example, you can enter a value of HOUR TO SECOND precision into a column defined as DAY TO SECOND. A value must always consist, however, of contiguous time units. In the previous example, you cannot enter only the HOUR and SECOND values; you must also include MINUTE values.

A valid INTERVAL literal contains the INTERVAL keyword, the values to be entered, and the field qualifiers. (See the discussion of literal intervals in the IBM Informix Guide to SQL: Syntax.) When a value contains only one field, the largest and smallest fields are the same.

When you enter a value in an INTERVAL column, you must specify the largest and smallest fields in the value, just as you do for DATETIME values. In addition, you can optionally specify the precision of the first field (and the scale of the last field if it is a FRACTION). If the largest and smallest field qualifiers are both FRACTION, you can specify only the scale in the last field.

Acceptable qualifiers for the largest and smallest fields are identical to the list of INTERVAL fields that [Table 2-5 on page 2-17](#) displays.

If you use the DB–Access TABLE menu, but you specify no INTERVAL field qualifiers, then a default INTERVAL qualifier, YEAR TO YEAR, is assigned.

The largest _qualifier_ in an INTERVAL value can be up to nine digits (except for FRACTION, which cannot be more than five digits), but if the value that you want to enter is greater than the default number of digits allowed for that field, you must explicitly identify the number of significant digits in the value that you enter. For example, to define an INTERVAL of DAY TO HOUR that can store up to 999 days, you could specify it the following way:

```
INTERVAL DAY(3) TO HOUR
```
INTERVAL literals use the same delimiters as DATETIME literals (except that MONTH and DAY time units are not valid within the same INTERVAL value). Table 2-6 shows the INTERVAL delimiters.

<table>
<thead>
<tr>
<th>Delimiter</th>
<th>Placement in an INTERVAL Literal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyphen</td>
<td>Between the YEAR and MONTH portions of the value</td>
</tr>
<tr>
<td>Blank space</td>
<td>Between the DAY and HOUR portions of the value</td>
</tr>
<tr>
<td>Colon</td>
<td>Between the HOUR, MINUTE, and SECOND portions of the value</td>
</tr>
<tr>
<td>Decimal point</td>
<td>Between the SECOND and FRACTION portions of the value</td>
</tr>
</tbody>
</table>

You can also enter INTERVAL values as character strings. The character string must include information for the same time units that were specified in the data-type declaration for the column. The INSERT statement in the following example shows an INTERVAL value entered as a character string:

```
INSERT INTO manufact (manu_code, manu_name, lead_time) VALUES ('800', 'Ball-Racquet Originals', '160')
```

Because the lead_time column is defined as INTERVAL DAY(3) TO DAY, this INTERVAL value requires only one field, the span of days required for lead time. If the character string does not contain information for all fields (or adds additional fields), the database server returns an error. For additional information on entering INTERVAL values as character strings, see the IBM Informix Guide to SQL: Syntax.

By default, all fields of an INTERVAL column are two-digit numbers, except for the year and fraction fields. The year field is stored as four digits. The fraction field requires n digits where 1 ≤ n ≤ 5, rounded up to an even number. You can use the following formula (rounded up to a whole number of bytes) to calculate the number of bytes required for an INTERVAL value:

\[
\text{total number of digits for all fields}/2 + 1
\]

For example, INTERVAL YEAR TO MONTH requires six digits (four for year and two for month), and requires 4, or (6/2) + 1, bytes of storage.

For information on using INTERVAL data in arithmetic and relational operations, see “Manipulating DATE with DATETIME and INTERVAL Values” on page 2-35. For information on using INTERVAL as a constant expression, see the description of the INTERVAL Field Qualifier in the IBM Informix Guide to SQL: Syntax.

**LIST(e) (IDS)**

The LIST data type is a collection type that stores ordered, non-unique elements; that is, it allows duplicate element values. The elements of a LIST have ordinal positions; that is, the list has a first element, a second element, and so on. (For a collection type with no ordinal positions, see “MULTISET(e) (IDS)” on page 2-21 and “SET(e) (IDS)” on page 2-27). No more than 97 columns of the same table can be declared as LIST data types. (The same restriction applies to SET and MULTISET collection types.)

By default, the database server inserts LIST elements at the end of the list. To support the ordinal position of a LIST, the INSERT statement provides the AT
clause. This clause allows you to specify the position at which you want to insert a list-element value. For more information, see the INSERT statement in the IBM Informix Guide to SQL: Syntax.

All elements in a LIST have the same element type. To specify the element type, use the following syntax:

```
LIST(element_type NOT NULL)
```

The `element_type` of a LIST can be any of the following data types:
- A built-in type, except SERIAL, SERIAL8, BYTE, and TEXT
- A distinct type
- An unnamed or named row type
- Another collection type
- An opaque type

You must specify the NOT NULL constraint for LIST elements. No other constraints are valid for LIST columns. For more information on the syntax of the LIST data type, see the IBM Informix Guide to SQL: Syntax.

You can use LIST where any other data type is valid. For example:
- After the IN predicate in the WHERE clause of a SELECT statement to search for matching LIST values
- As an argument to the CARDINALITY or `mi_collection_card()` function to determine the number of elements in a LIST column

You cannot use LIST values as arguments to an aggregate function such as AVG, MAX, MIN, or SUM.

Two list values are equal if they have the same elements in the same order. The following examples both are list values but are not equal:

```
LIST("blue", "green", "yellow")
LIST("yellow", "blue", "green")
```

The above statements are not equal because the values are not in the same order. To be equal, the second statement would have to be:

```
LIST("blue", "green", "yellow")
```

**LVARCHAR(m) (IDS)**

You can use the LVARCHAR data type to create a column for storing variable-length character strings whose upper limit (m) can be up to 32,739 bytes. (You can use the VARCHAR data type for strings no longer than 255 bytes.)

By default, the database server interprets quoted strings as LVARCHAR types. It also uses LVARCHAR for input and output casts for opaque data types.

The LVARCHAR data type stores opaque data types in the string (external) format. Each opaque type has an input support function and cast, which convert it from LVARCHAR to a form that database servers can manipulate. Each opaque type also has an output support function and cast, which convert it from its internal representation to LVARCHAR.

**Important:** When LVARCHAR is declared (with no size specification) as the data type of a column in a database table, the default maximum size is 2
kilobytes (2048 bytes), but you can specify an explicit maximum length of up to 32,739 bytes. When LVARCHAR is used in I/O operations on an opaque data type, however, the maximum size is limited only by the operating system.

LVARCHAR is implemented as a built-in opaque UDT. Only a subset of the string operations on CHAR and VARCHAR values are valid for LVARCHAR, and like other opaque types, LVARCHAR columns of remote tables are not accessible in distributed queries. For more information about LVARCHAR, see IBM Informix User-Defined Routines and Data Types Developer’s Guide.

MONEY(p,s)

The MONEY data type stores currency amounts. Like the DECIMAL(p,s) data type, MONEY can store fixed-point numbers up to a maximum of 32 significant digits, where p is the total number of significant digits (the precision) and s is the number of digits to the right of the decimal point (the scale).

Unlike the DECIMAL data type, the MONEY data type is always treated as a fixed-point decimal number. The database server defines the data type MONEY(p) as DECIMAL(p,2). If the precision and scale are not specified, the database server defines a MONEY column as DECIMAL(16,2).

You can use the following formula (rounded down to a whole number of bytes) to calculate the byte storage for a MONEY data type:

If the scale is odd: \( N = \frac{\text{precision} + 4}{2} \)
If the scale is even: \( N = \frac{\text{precision} + 3}{2} \)

For example, a MONEY data type with a precision of 16 and a scale of 2 (MONEY(16,2)) requires 10 or \((16 + 3)/2\), bytes of storage.

In the default locale, client applications format values from MONEY columns with the following currency notation:

- A currency symbol: a dollar sign (\$) at the front of the value
- A thousands separator: a comma (,) that separates every three digits in the integer part of the value
- A decimal point: a period (.) between the integer and fractional parts of the value

To change the format for MONEY values, change the DBMONEY environment variable. For valid DBMONEY settings, see “DBMONEY” on page 3-24.

The default value that the database server uses for scale is locale-dependent. The default locale specifies a default scale of two. For non-default locales, if the scale is omitted from the declaration, the database server creates MONEY values with a locale-specific scale.

The currency notation that client applications use is locale-dependent. If you specify a nondefault locale, the client uses a culture-specific format for MONEY values that might differ from the default U.S. English format in the leading (or trailing) currency symbol, thousands separator, and decimal separator, depending on what the locale files specify. For more information on locale dependency, see the IBM Informix GLS User’s Guide.
MULTISET(e) (IDS)

The MULTISET data type is a collection type that stores a non-ordered set that can include duplicate element values. The elements in a MULTISET have no ordinal position. That is, there is no concept of a first, second, or third element in a MULTISET. (For a collection type with ordinal positions for elements, see "LIST(e) (IDS)" on page 2-18)

All elements in a MULTISET have the same element type. To specify the element type, use the following syntax:

MULTISET(element_type NOT NULL)

The element_type of a collection can be any of the following types:

- Any built-in type, except SERIAL, SERIAL8, BYTE, and TEXT
- An unnamed or a named ROW type
- Another collection type or opaque type

You can use MULTISET anywhere that you use any other data type, unless otherwise indicated. For example:

- After the IN predicate in the WHERE clause of a SELECT statement to search for matching MULTISET values
- As an argument to the CARDINALITY or mi_collection_card() function to determine the number of elements in a MULTISET column

You cannot use MULTISET values as arguments to an aggregate function such as AVG, MAX, MIN, or SUM.

You must specify the NOT NULL constraint for MULTISET elements. No other constraints are valid for MULTISET columns. For more information on the MULTISET collection type, see the IBM Informix Guide to SQL: Syntax.

Two multiset data values are equal if they have the same elements, even if the elements are in different positions within the set. The following examples are both multiset values but are not equal:

MULTISET ("blue", "green", "yellow")
MULTISET ("blue", "green", "yellow", "blue")

The following multiset values are equal:

MULTISET ("blue", "green", "blue", "yellow")
MULTISET ("blue", "green", "yellow", "blue")

No more than 97 columns of the same table can be declared as MULTISET data types. (The same restriction applies to SET and LIST collection types.)

Named ROW

See "ROW, Named (IDS)" on page 2-23

NCHAR(n)

The NCHAR data type stores fixed-length character data. The data can be a string of single-byte or multibyte letters, digits, and other symbols that are supported by the code set of the database locale. The main difference between CHAR and NCHAR data types is the collating order.
The collation order of the CHAR data type follows the code-set order, but the collating order of the NCHAR data type can be a localized order, if DB_LOCALE (or SET COLLATION) specifies a localized collation. For more information about NCHAR, see the description of "DBNLS (IDS)" on page 3-25.

**NUMERIC(p,s)**

The NUMERIC data type is a synonym for fixed-point DECIMAL.

**NVARCHAR(m,r)**

The NVARCHAR data type stores strings of varying lengths. The string can include digits, symbols, and single-byte and (in some locales) multibyte characters. The main difference between VARCHAR and NVARCHAR data types is the collation order. Collation of VARCHAR data follows code-set order, but NVARCHAR collation can be locale specific, if DB_LOCALE (or SET COLLATION) has specified a localized collation. (The section "Collating VARCHAR Values" on page 2-31 describes an exception.)

A column declared as NVARCHAR, without parentheses or parameters, has a maximum size of one byte, and a reserved size of zero.

No more than 195 columns of the same table can be NVARCHAR data types.

**Opaque (IDS)**

An opaque type is a data type for which you must provide the following information to the database server:

- A data structure for how the data values are stored on disk
- Support functions to determine how to convert between the disk storage format and the user format for data entry and display
- Secondary access methods that determine how the index on this data type is built, used, and manipulated
- User functions that use the data type
- A system catalog entry to register the opaque type in the database

The internal structure of an opaque type is not visible to the database server and can only be accessed through user-defined routines. Definitions for opaque types are stored in the sysxtypes system catalog table. These SQL statements maintain the definitions of opaque types in the database:

- The CREATE OPAQUE TYPE statement registers a new opaque type in the database.
- The DROP TYPE statement removes a previously defined opaque type from the database.

For more information on the above-mentioned SQL statements, see the IBM Informix Guide to SQL: Syntax. For information on how to create opaque types and an example of an opaque type, see IBM Informix User-Defined Routines and Data Types Developer’s Guide.

**REAL**

The REAL data type is a synonym for SMALLFLOAT.
ROW, Named (IDS)

A named ROW type is declared by its name. That identifier must be unique within the schema. An unnamed ROW type is a ROW type that contains fields but has no user-defined name. Only named ROW types support data type inheritance. For more information, see “ROW Data Types” on page 2-40.

Defining Named ROW Types

You must declare and register in the database a new named ROW type by using the CREATE ROW TYPE statement of SQL. Definitions for named ROW types are stored in the sysxtdtypes system catalog table.

The fields of a ROW data type can be any built-in data type or UDT, but TEXT or BYTE fields of a ROW type are valid in typed tables only. If you want to assign a ROW type to a column, its elements cannot be TEXT or BYTE data types.

In general, the data type of a field of a ROW type can be any of these types:

- A built-in type (except for the TEXT or BYTE data types)
- A collection type (LIST, MULTISET, or SET)
- A distinct type
- Another named or unnamed ROW type
- An opaque type

These SQL statements maintain the definitions of named ROW data types:

- The CREATE ROW TYPE statement adds a named ROW type to the database.
- The DROP ROW TYPE statement removes a previously defined named ROW type from the database.

No more than 195 columns of the same table can be named ROW types.

For details about these SQL syntax statements, see the IBM Informix Guide to SQL: Syntax. For examples of how to create and use named ROW types, see the IBM Informix Database Design and Implementation Guide.

Equivalence and Named ROW Types

No two named ROW types can be equal, even if they have identical structures, because they have different names. For example, the following named ROW types have the same structure (the same number of fields and the same order of data types of fields within the row) but are not equal:

```
name_t (lname CHAR(15), initial CHAR(1), fname CHAR(15))
emp_T (lname CHAR(15), initial CHAR(1), fname CHAR(15))
```

Named ROW Types and Inheritance

Named ROW types can be part of a type-inheritance hierarchy. One named ROW type can be the parent (or supertype) of another named ROW type. A subtype in a hierarchy inherits all the properties of its supertype. Type inheritance is discussed in the CREATE ROW TYPE statement in the IBM Informix Guide to SQL: Syntax and in the IBM Informix Database Design and Implementation Guide.

Typed Tables

Tables that are part of an inheritance hierarchy must be typed tables. Typed tables are tables that have been assigned a named ROW type. For the syntax you use to create typed tables, see the CREATE TABLE statement in the IBM Informix Guide to SQL: Syntax. Table inheritance and its relation to type inheritance is also discussed.
ROW, Unnamed (IDS)

An unnamed ROW type contains fields but has no user-declared name. An unnamed ROW type is defined by its structure. Two unnamed ROW types are equal if they have the same structure (meaning the ordered list of the data types of the fields). If two unnamed ROW types have the same number of fields, and if the order of the data type of each field in one ROW type matches the order of data types of the corresponding fields in the other ROW data type, then the two unnamed ROW data types are equal.

For example, the following unnamed ROW types are equal:
ROW (lname char(15), initial char(1) fname char(15))
ROW (dept char(15), rating char(1) name char(15))

The following ROW types have the same number of fields and the same data types, but are not equal, because their fields are not in the same order:
ROW (x integer, y varchar(20), z real)
ROW (x integer, z real, y varchar(20))

A field of an unnamed ROW type can be any of the following data types:
• A built-in type
• A collection type
• A distinct type
• Another ROW type
• An opaque type

Unnamed ROW types cannot be used in typed tables or in type inheritance hierarchies. For more information on unnamed ROW types, see the IBM Informix Guide to SQL: Syntax and the IBM Informix Database Design and Implementation Guide.

Creating Unnamed ROW Types
You can create an unnamed ROW type in several ways:
• You can declare an unnamed ROW type using the ROW keyword. Each field in a ROW can have a different field type. To specify the field type, use the following syntax:
  ROW(field_name field_type, ...)
  The field_name must conform to the rules for SQL identifiers. (See the Identifier section in the IBM Informix Guide to SQL: Syntax.)
• To generate an unnamed ROW type, use the ROW keyword as a constructor with a series of values. A corresponding unnamed ROW type is created, using the default data types of the specified values.
  For example, the following declaration:
  ROW(1, 'abc', 5.30)
  defines this unnamed ROW data type:
  ROW (x INTEGER, y VARCHAR, z DECIMAL)
• You can create an unnamed ROW type by an implicit or explicit cast from a named ROW type or from another unnamed ROW type.
• The rows of any table (except a table defined on a named ROW type) are unnamed ROW types.
No more than 195 columns of the same table can be unnamed ROW types.

**Inserting Values into Unnamed ROW Type Columns**

When you specify field values for an unnamed ROW type, list the field values after the constructor and between parentheses. For example, suppose you have an unnamed ROW-type column. The following INSERT statement adds one group of field values to this ROW column:

```
INSERT INTO table1 VALUES (ROW(4, 'abc'))
```

You can specify a ROW column in the IN predicate in the WHERE clause of a SELECT statement to search for matching ROW values. For more information, see the Condition section in the *IBM Informix Guide to SQL: Syntax*.

**SERIAL(n)**

The SERIAL data type stores a sequential integer, in the positive range of the INT8 data type, that is automatically assigned by the database server when a new row is inserted. A table can have no more than one SERIAL column, but it can have one SERIAL and one SERIAL8 column.

SERIAL values in a column are not automatically unique. You must apply a unique index or primary key constraint to this column to prevent duplicate serial numbers. If you use the interactive schema editor in DB–Access to define the table, a unique index is applied automatically to a SERIAL column.

SERIAL numbers might not be consecutive, because of concurrent users, rollbacks, and other factors.

The DEFINE variable LIKE column syntax of SPL for indirect typing declares a variable of the INTEGER data type if column is a SERIAL data type.

The default serial starting number is 1, but you can assign a non-default initial value, $n$, when you create or alter the table. Any number greater than 0 can be your starting number. The maximum SERIAL is 2,147,483,647. If you assign a number greater than 2,147,483,647, you receive a syntax error. (Use the SERIAL8 data type, rather than SERIAL, if you need a larger range.)

After a nonzero number is assigned, it cannot be changed. You can insert a value into a SERIAL column (using the INSERT statement) or reset a serial column (using the ALTER TABLE statement), if the new value does not duplicate any existing value in the column. To insert into a SERIAL column, your database server increments by one the previous value (or the reset value, if that is larger) and assigns the result as the entered value. If ALTER TABLE has reset the next value of a SERIAL column to a value smaller than values already in that column, however, the next value follows this formula:

$$(\text{maximum existing value in SERIAL column}) + 1$$

For example, if you reset the serial value of `customer.customer_num` to 50, when the largest existing value is 128, the next assigned number will be 129. For more details on SERIAL data entry, see the *IBM Informix Guide to SQL: Syntax*.

A SERIAL column can store unique codes (for example, order, invoice, or customer numbers). SERIAL data values require four bytes of storage, and have the same precision as the INTEGER data type. For details of another way to assign unique whole numbers to each row of a database table, see the CREATE SEQUENCE statement in *IBM Informix Guide to SQL: Syntax*. 
SERIAL8(n)

The SERIAL8 data type stores a sequential integer, in the positive range of the INT8 data type, that is assigned automatically by the database server when a new row is inserted. It behaves like the SERIAL data type, but with a larger range. (For more information on how to insert values into SERIAL8 columns, see the IBM Informix Guide to SQL: Syntax.)

A SERIAL8 data column is commonly used to store large, unique numeric codes (for example, order, invoice, or customer numbers). SERIAL8 data values have the same precision and storage requirements as INT8 values (page 2-16). The following restrictions apply to SERIAL8 columns:

- You can define only one SERIAL8 column in a table.
  A table, however, can have one SERIAL8 and one SERIAL column.
- SERIAL8 column values are not automatically unique.
  You must apply a unique index or primary key constraint to this column to prevent duplicate SERIAL8 numbers.
- The SERIAL8 data type does not allow a negative, zero, or NULL value.

The DEFINE variable LIKE column syntax of SPL for indirect typing declares a variable of the INT8 data type if column is a SERIAL8 data type.

Assigning a Starting Value for SERIAL8

The default serial starting number is 1, but you can assign an initial value, n, when you create or alter the table. To start the values at 1 in a SERIAL8 column of a table, give the value 0 for the SERIAL8 column when you insert rows into that table. The database server will assign the value 1 to the SERIAL8 column of the first row of the table. The largest SERIAL8 value that you can assign is \(2^{63} - 1\) (9,223,372,036,854,775,807). If you assign a value greater than this, you receive a syntax error. When the database server generates a SERIAL8 value of this maximum number, it wraps around and starts generating values beginning at 1.

After a nonzero SERIAL8 number is assigned, it cannot be changed. You can, however, insert a value into a SERIAL8 column (using the INSERT statement) or reset the SERIAL8 value n (using the ALTER TABLE statement), as long as that value does not duplicate any existing values in the column.

When you insert a number into a SERIAL8 column or reset the next value of a SERIAL8 column, your database server assigns the next number in sequence to the number entered. If you reset the next value of a SERIAL8 column to a value that is less than the values already in that column, however, the next value is computed using the following formula:

\[ \text{maximum existing value in SERIAL8 column} + 1 \]

For example, if you reset the SERIAL8 value of the customer_num column in the customer table to 50, when the highest-assigned customer number is 128, the next customer number assigned is 129.

Using SERIAL8 with INT8

All the arithmetic operators that are valid for INT8 (such as +, -, *, and /) and all the SQL functions that are valid for INT8 (such as ABS, MOD, POW, and so on) are also valid for SERIAL8 values. Data conversion rules that apply to INT8 also apply to SERIAL8, but with a NOT NULL constraint on SERIAL8.
The value of a SERIAL8 column of one table can be stored in an INT8 columns of another table. In the second table, however, the INT8 values are not subject to the constraints on the original SERIAL8 column.

**SET(e) (IDS)**

The SET data type is an unordered collection type that stores unique elements; duplicate element values are not valid as explained in *IBM Informix Guide to SQL: Syntax*. (For a collection type that supports duplicate values, see the description of MULTISET in *“MULTISET(e) (IDS)” on page 2-21*).

No more than 97 columns of the same table can be declared as SET data types. (The same restriction also applies to MULTISET and LIST collection types.)

The elements in a SET have no ordinal position. That is, no construct of a first, second, or third element in a SET exists. (For a collection type with ordinal positions for elements, see *“LIST(e) (IDS)” on page 2-18*.) All elements in a SET have the same element type. To specify the element type, use this syntax:

```
SET(element_type NOT NULL)
```

The `element_type` of a collection can be any of the following types:
- A built-in type, except SERIAL, SERIAL8, BYTE, and TEXT
- A named or unnamed ROW type
- Another collection type
- An opaque type

You must specify the NOT NULL constraint for SET elements. No other constraints are valid for SET columns. For more information on the syntax of the SET collection type, see the *IBM Informix Guide to SQL: Syntax*.

You can use SET anywhere that you use any other data type, unless otherwise indicated. For example:
- After the IN predicate in the WHERE clause of a SELECT statement to search for matching SET values
- As an argument to the CARDINALITY or `mi_collection_card()` function to determine the number of elements in a SET column

SET values are not valid as arguments to an aggregate function such as AVG, MAX, MIN, or SUM. For more information, see the Condition and Expression sections in the *IBM Informix Guide to SQL: Syntax*.

The following examples declare two sets. The first statement declares a set of integers and the second declares a set of character elements.

```
SET(INTEGER NOT NULL)
SET(CHAR(20) NOT NULL)
```

The following examples construct the same sets from value lists:

```
SET {1, 5, 13}
SET {"Oakland", "Menlo Park", "Portland", "Lenexa"}
```

In the following example, a SET constructor function is part of a CREATE TABLE statement:
CREATE TABLE tab
(
    c CHAR(5),
    s SET(INTEGER NOT NULL)
);

The following set values are equal:
SET("blue", "green", "yellow")
SET("yellow", "blue", "green")

**SMALLFLOAT**

The SMALLFLOAT data type stores single-precision floating-point numbers with approximately nine significant digits. SMALLFLOAT corresponds to the float data type in C. The range of values for a SMALLFLOAT data type is the same as the range of values for the C float data type on your computer.

A SMALLFLOAT data type column typically stores scientific numbers that can be calculated only approximately. Because floating-point numbers retain only their most significant digits, the number that you enter in this type of column and the number the database displays might differ slightly depending on how your computer stores floating-point numbers internally.

For example, you might enter a value of 1.1000001 in a SMALLFLOAT field and, after processing the SQL statement, the application might display this value as 1.1. This difference occurs when a value has more digits than the floating-point number can store. In this case, the value is stored in its approximate form with the least significant digits treated as zeros.

SMALLFLOAT data types usually require 4 bytes of storage. Conversion of a SMALLFLOAT value to a DECIMAL value results in 9 digits of precision.

**SMALLINT**

The SMALLINT data type stores small whole numbers that range from -32,767 to 32,767. The maximum negative number, -32,768, is a reserved value and cannot be used. The SMALLINT value is stored as a signed binary integer.

Integer columns typically store counts, quantities, and so on. Because the SMALLINT data type requires only two bytes per value, arithmetic operations are performed efficiently. SMALLINT, however, stores only a limited range of values, compared to other built-in numeric data types. If a number is outside the range of the minimum and maximum SMALLINT values, the database server does not store the data value, but instead issues an error message.

**TEXT**

The TEXT data type stores any kind of text data. It can contain both single-byte and multibyte characters that the locale supports. The term *simple large object* refers to the TEXT and BYTE data types.

A TEXT column has a theoretical limit of $2^{31}$ bytes (two gigabytes) and a practical limit that your available disk storage determines.

**Important:** An error results if you try to return a TEXT column from a subquery, even if no TEXT column is used in a comparison condition or with the IN predicate.
No more than 195 columns of the same table can be declared as TEXT data types. (The same restriction also applies to BYTE data types.)

You can store, retrieve, update, or delete the values in a TEXT column. You cannot, however, use TEXT operands in arithmetic or string expressions, nor can you assign literals to TEXT columns in the SET clause of the UPDATE statement. You also cannot use TEXT values in any of the following ways:

- With aggregate functions
- With the IN clause
- With the MATCHES or LIKE clauses
- With the GROUP BY clause
- With the ORDER BY clause

You can use TEXT operands in Boolean expressions only when you are testing for NULL values with the IS NULL or IS NOT NULL operators.

You can insert data into TEXT columns in the following ways:

- With the dbload or onload utilities
- With the LOAD statement (DB–Access)
- From TEXT host variables (ESQL)

You cannot use a quoted text string, number, or any other actual value to insert or update TEXT columns.

When you select a TEXT column, you can choose to receive all or part of it. To retrieve it all, use the regular syntax for selecting a column. You can also select any part of a TEXT column by using subscripts, as this example shows:

```
SELECT cat_descr [1,75] FROM catalog WHERE catalog_num = 10001
```

This statement reads the first 75 bytes of the cat_desc column associated with the catalog_num value 10001.

A built-in cast exists to convert TEXT objects to CLOB objects. For more information, see the IBM Informix Database Design and Implementation Guide.

Strings of the TEXT data type are collated in code-set order. For more information on collating orders, see the IBM Informix GLS User’s Guide.

**Nonprintable Characters in TEXT Values**

TEXT columns typically store documents, program source files, and so on. In the default U.S. English locale, data objects of type TEXT can contain a combination of printable ASCII characters and the following control characters:

- Tab (CTRL-I)
- New line (CTRL-J)
- New page (CTRL-L)

Both printable and nonprintable characters can be inserted in text columns. IBM Informix products do not do any checking of data values that are inserted in a column of the TEXT data type. (Applications may have difficulty, however, in displaying TEXT values that include non-printable characters.) For detailed information on entering and displaying nonprintable characters, refer to “Nonprintable Characters with CHAR” on page 2-7.
**Unnamed ROW**

See “ROW, Unnamed (IDS)” on page 2-24

**VARCHAR(m,r)**

The VARCHAR data type stores character strings of varying length that contain single-byte and (if the locale supports them) multibyte characters, where \( m \) is the maximum size (in bytes) of the column and \( r \) is the minimum number of bytes reserved for that column. A column declared as VARCHAR without parentheses or parameters has a maximum size of one byte, and a reserved size of zero.

The VARCHAR data type is the Informix implementation of a character varying data type. The ANSI standard data type for varying-length character strings is CHARACTER VARYING.

The size of the maximum size \( (m) \) parameter of a VARCHAR column can range from 1 to 255 bytes. If you are placing an index on a VARCHAR column, the maximum size is 254 bytes. You can store character strings that are shorter, but not longer, than the \( m \) value that you specify.

Specifying the minimum reserved space \( (r) \) parameter is optional. This value can range from 0 to 255 bytes but must be less than the maximum size \( (m) \) of the VARCHAR column. If you do not specify any minimum value, it defaults to 0. You should specify this parameter when you initially intend to insert rows with short or NULL character strings in the column but later expect the data to be updated with longer values.

For variable-length strings longer than 255 bytes, you can use the LVARCHAR data type, whose upper limit is 32,739 bytes, instead of VARCHAR. Because LVARCHAR is implemented as a built-in opaque data type, however, you cannot access LVARCHAR columns in distributed queries of remote tables.

In an index based on a VARCHAR column (or on a NVARCHAR column), each index key has a length that is based on the data values that are actually entered, rather than on the declared maximum size of the column. (See, however, “IFX_PAD_VARCHAR (IDS)” on page 3-44 for information on how you can configure the effective size of VARCHAR and NVARCHAR data strings that Dynamic Server sends or receives.)

When you store a string in an VARCHAR column, only the actual data characters are stored. The database server does not strip a VARCHAR string of any user-entered trailing blanks, nor pad a VARCHAR value to the declared length of the column. If you specify a reserved space \( (r) \), but some data strings are shorter than \( r \) bytes, some space reserved for rows goes unused.

VARCHAR values are compared to other VARCHAR values (and to other character-string data types) in the same way that CHAR values are compared. The shorter value is padded on the right with blank spaces until the values have equal lengths; then they are compared for the full length.

No more than 195 columns of the same table can be VARCHAR data types.

**Nonprintable Characters with VARCHAR**

Nonprintable VARCHAR characters are entered, displayed, and treated in the same way that nonprintable characters in CHAR values are treated. For details, see the section “Nonprintable Characters with CHAR” on page 2-7
Storing Numeric Values in a VARCHAR Column
When you insert a numeric value in a VARCHAR column, the stored value does not get padded with trailing blanks to the maximum length of the column. The number of digits in a numeric VARCHAR value is the number of characters that you need to store that value. For example, in the next example, the value stored in table mytab is 1.
create table mytab (col1 varchar(10));
insert into mytab values (1);

Tip: VARCHAR treats C null (binary 0) and string terminators as termination characters for nonprintable characters.

Multibyte Characters with VARCHAR
In some East Asian locales, VARCHAR data types can store multibyte characters if the database locale supports a multibyte code set. If you store multibyte characters, make sure to calculate the number of bytes needed. For more information, see the IBM Informix GLS User’s Guide.

Collating VARCHAR Values
The main difference between the NVARCHAR and the VARCHAR data types (like the difference between CHAR and NCHAR) is the difference in collating order. In general, collation of VARCHAR (like CHAR and LVARCHAR ) values is in the order of the characters as they appear in the code set.

An exception is the MATCHES operator, which applies a localized collation to NVARCHAR and VARCHAR values (as well as to CHAR, LVARCHAR, and NCHAR values) if you use bracket ( [ ] ) symbols to define ranges when DB_LOCALE (or SET COLLATION) has specified a localized collating order. For more information, see the IBM Informix GLS User’s Guide.

Built-In Data Types
Informix database servers support the following built-in data types.

<table>
<thead>
<tr>
<th>Category</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>CHAR, CHARACTER VARYING, LVARCHAR,</td>
</tr>
<tr>
<td></td>
<td>NCHAR, NVARCHAR, VARCHAR,</td>
</tr>
<tr>
<td></td>
<td>IDSSECURITYLABEL</td>
</tr>
<tr>
<td>Numeric</td>
<td>DECIMAL, FLOAT, INT8, INTEGER,</td>
</tr>
<tr>
<td></td>
<td>MONEY, SERIAL, SERIAL8,</td>
</tr>
<tr>
<td></td>
<td>SMALLFLOAT, SMALLINT</td>
</tr>
<tr>
<td>Large-object</td>
<td>Simple-large-object types: BYTE,</td>
</tr>
<tr>
<td></td>
<td>TEXT</td>
</tr>
<tr>
<td></td>
<td>Smart-large-object types: BLOB,</td>
</tr>
<tr>
<td></td>
<td>CLOB</td>
</tr>
<tr>
<td>Time</td>
<td>DATE, DATETIME, INTERVAL</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>BOOLEAN</td>
</tr>
</tbody>
</table>

Extended Parallel Server does not support BLOB, CLOB, or LVARCHAR. For a description of character, numeric, and miscellaneous data types, refer to the appropriate entry in “Description of Data Types” on page 2-5. Page references are in the alphabetical list in Table 2-1 on page 2-3.

Sections that follow provide additional information on large-object and time data types.
Large-Object Data Types

A large object is a data object that is logically stored in a table column but physically stored independent of the column. Large objects are stored separate from the table because they typically store a large amount of data. Separation of this data from the table can increase performance.

Figure 2-4 shows the large-object data types.

![Large-Object Data Types Diagram]

**Figure 2-4. Large-Object Data Types**

Only Dynamic Server supports BLOB and CLOB data types.

For the relative advantages and disadvantages of simple and smart large objects, see the *IBM Informix Database Design and Implementation Guide*.

Simple Large Objects

Simple large objects are a category of large objects that have a theoretical size limit of $2^{31}$ bytes and a practical limit that your disk capacity determines. Informix database servers support these simple-large-object data types:

- **BYTE** Stores binary data. For more detailed information about this data type, see the description on page 2-6.
- **TEXT** Stores text data. For more detailed information about this data type, see the description on page 2-28.

No more than 195 columns of the same table can be declared as BYTE or TEXT data types. Unlike smart large objects, simple large objects do not support random access to the data. When you transfer a simple large object between a client application and the database server, you must transfer the entire BYTE or TEXT value. If the data cannot fit into memory, you must store the data value in an operating-system file and then retrieve it from that file.

The database server stores simple large objects in *blobspaces*. A *blobspace* is a logical storage area that contains one or more chunks that only store BYTE and TEXT data. For information on how to define blobspaces, see your *IBM Informix Administrator’s Guide*.

Smart Large Objects (IDS)

Smart large objects are a category of large objects that support random access to the data and are generally recoverable. The random access feature allows you to seek and read through the smart large object as if it were an operating-system file.

Smart large objects are also useful for opaque data types with large storage requirements. (See the description of opaque data types in "Opaque Data Types" on page 2-40.) They have a theoretical size limit of $2^{62}$ bytes and a practical limit that your disk capacity determines.

Dynamic Server supports the following smart-large-object data types:
BLOB Stores binary data. For more information about this data type, see the description on page 2-5.

CLOB Stores text data. For more information about this data type, see 2-8.

Dynamic Server stores smart large objects in sbspaces. An sbspace is a logical storage area that contains one or more chunks that store only BLOB and CLOB data. For information on how to define sbspaces, see your IBM Informix Performance Guide.

When you define a BLOB or CLOB column, you can determine the following large-object characteristics:

- LOG and NOLOG: whether the database server should log the smart large object in accordance with the current database log mode
- KEEP ACCESS TIME and NO KEEP ACCESS TIME: whether the database server should keep track of the last time the smart large object was accessed
- HIGH INTEG and MODERATE INTEG: whether the database server should use page headers to detect data corruption

Use of these characteristics can affect performance. For information, see your IBM Informix Performance Guide.

When an SQL statement accesses a smart-large-object, the database server does not send the actual BLOB or CLOB data. Instead, it establishes a pointer to the data and returns this pointer. The client application can then use this pointer in open, read, or write operations on the smart large object.

To access a BLOB or CLOB column from within a client application, use one of the following application programming interfaces (APIs):

- From within an IBM Informix ESQL/C program, use the smart-large-object API. (For more information, see the IBM Informix ESQL/C Programmer’s Manual.)
- From within a DataBlade module, use the Client and Server API. (For more information, see the IBM Informix DataBlade API Programmer’s Guide.)

For information on smart large objects, see the IBM Informix Guide to SQL: Syntax and IBM Informix Database Design and Implementation Guide.

**Time Data Types**

DATE and DATETIME data values represent zero-dimensional points in time; INTERVAL data values represent 1-dimensional spans of time, with positive or negative values. DATE precision is always an integer count of days, but various field qualifiers can define the DATETIME and INTERVAL precision. You can use DATE, DATETIME, and INTERVAL data in arithmetic and relational expressions. You can manipulate a DATETIME value with another DATETIME value, an INTERVAL value, the current time (specified by the keyword CURRENT), or some unit of time (using the keyword UNITS).

You can use a DATE value in most contexts where a DATETIME value is valid, and vice versa. You also can use an INTERVAL operand in arithmetic operations where a DATETIME value is valid. In addition, you can add two INTERVAL values and multiply or divide an INTERVAL value by a number.

An INTERVAL column can hold a value that represents the difference between two DATETIME values or the difference between (or sum of) two INTERVAL values. In either case, the result is a span of time, which is an INTERVAL value. Conversely,
if you add or subtract an INTERVAL from a DATETIME value, another DATETIME value is produced, because the result is a specific time.

Table 2-7 lists the binary arithmetic operations that you can perform on DATE, DATETIME, and INTERVAL operands, as well as the data type that is returned by the arithmetic expression.

Table 2-7. Arithmetic Operations on DATE, DATETIME, and INTERVAL Values

<table>
<thead>
<tr>
<th>Operand 1</th>
<th>Operator</th>
<th>Operand 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>-</td>
<td>DATETIME</td>
<td>INTERVAL</td>
</tr>
<tr>
<td>DATETIME</td>
<td>-</td>
<td>DATE</td>
<td>INTERVAL</td>
</tr>
<tr>
<td>DATE</td>
<td>+ or -</td>
<td>INTERVAL</td>
<td>DATETIME</td>
</tr>
<tr>
<td>DATETIME</td>
<td>-</td>
<td>DATETIME</td>
<td>INTERVAL</td>
</tr>
<tr>
<td>DATETIME</td>
<td>+ or -</td>
<td>INTERVAL</td>
<td>DATETIME</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>+</td>
<td>DATETIME</td>
<td>DATETIME</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>+ or -</td>
<td>INTERVAL</td>
<td>INTERVAL</td>
</tr>
<tr>
<td>DATETIME</td>
<td>-</td>
<td>CURRENT</td>
<td>INTERVAL</td>
</tr>
<tr>
<td>CURRENT</td>
<td>-</td>
<td>DATETIME</td>
<td>INTERVAL</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>+</td>
<td>CURRENT</td>
<td>DATETIME</td>
</tr>
<tr>
<td>CURRENT</td>
<td>+ or -</td>
<td>INTERVAL</td>
<td>DATETIME</td>
</tr>
<tr>
<td>DATETIME</td>
<td>+ or -</td>
<td>UNITS</td>
<td>DATETIME</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>+ or -</td>
<td>UNITS</td>
<td>INTERVAL</td>
</tr>
<tr>
<td>INTERVAL</td>
<td>* or /</td>
<td>NUMBER</td>
<td>INTERVAL</td>
</tr>
</tbody>
</table>

No other combinations are allowed. You cannot add two DATETIME values because this operation does not produce either a specific time or a span of time. For example, you cannot add December 25 and January 1, but you can subtract one from the other to find the time span between them.

**Manipulating DATETIME Values**

You can subtract most DATETIME values from each other. Dates can be in any order and the result is either a positive or a negative INTERVAL value. The first DATETIME value determines the precision of the result, which includes the same time units as the first operand.

If the second DATETIME value has fewer fields than the first, the precision of the second operand is increased automatically to match the first.

In the following example, subtracting the DATETIME YEAR TO HOUR value from the DATETIME YEAR TO MINUTE value results in a positive interval value of 60 days, 1 hour, and 30 minutes. Because minutes were not included in the second operand, the database server sets the minutes value for the second operand to 0 before performing the subtraction.

**Example:**

```
DATETIME (2003-9-30 12:30) YEAR TO MINUTE
  - DATETIME (2003-8-1 11) YEAR TO HOUR
```

**Result:** INTERVAL (60 01:30) DAY TO MINUTE
If the second DATETIME operand has more fields than the first (regardless of whether the precision of the extra fields is larger or smaller than those in the first operand), the additional time unit fields in the second value are ignored in the calculation.

In the next expression (and its result), the year is not included for the second operand. Therefore, the year is set automatically to the current year (from the system clock-calendar), in this example 2005, and the resulting INTERVAL is negative, which indicates that the second date is later than the first.

\[
\text{DATETIME (2005-9-30) YEAR TO DAY} \\
- \text{DATETIME (10-1) MONTH TO DAY}
\]

Result: INTERVAL (-1) DAY TO DAY [assuming that the current year is 2005]

**Manipulating DATETIME with INTERVAL Values**

INTERVAL values can be added to or subtracted from DATETIME values. In either case, the result is a DATETIME value. If you are adding an INTERVAL value to a DATETIME value, the order of values is unimportant; however, if you are subtracting, the DATETIME value must come first. Adding or subtracting a positive INTERVAL value simply moves the DATETIME result forward or backward in time. The expression shown in the following example moves the date ahead by three years and five months:

\[
\text{DATETIME (2000-8-1) YEAR TO DAY} \\
+ \text{INTERVAL (3-5) YEAR TO MONTH}
\]

Result: DATETIME (2004-01-01) YEAR TO DAY

**Important:** Evaluate the logic of your addition or subtraction. Remember that months can have 28, 29, 30, or 31 days and that years can have 365 or 366 days.

In most situations, the database server automatically adjusts the calculation when the operands do not have the same precision. In certain contexts, however, you must explicitly adjust the precision of one value to perform the calculation. If the INTERVAL value you are adding or subtracting has fields that are not included in the DATETIME value, you must use the EXTEND function to increase the precision of the DATETIME value. (For more information on the EXTEND function, see the Expression segment in the *IBM Informix Guide to SQL: Syntax*.)

For example, you cannot subtract an INTERVAL MINUTE TO MINUTE value from the DATETIME value in the previous example that has a YEAR TO DAY field qualifier. You can, however, use the EXTEND function to perform this calculation, as the following example shows:

\[
\text{EXTEND (DATETIME (2008-8-1) YEAR TO DAY, YEAR TO MINUTE)} \\
- \text{INTERVAL (720) MINUTE(3) TO MINUTE}
\]

Result: DATETIME (2008-07-31 12:00) YEAR TO MINUTE

The EXTEND function allows you to explicitly increase the DATETIME precision from YEAR TO DAY to YEAR TO MINUTE. This allows the database server to perform the calculation, with the resulting extended precision of YEAR TO MINUTE.

**Manipulating DATE with DATETIME and INTERVAL Values**

You can use DATE operands in some arithmetic expressions with DATETIME or INTERVAL operands by writing expressions to do the manipulating, as shown in Table 2-6 on page 2-36.
Table 2-8. Results of Expressions That Manipulate DATE with DATETIME or INTERVAL Values

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE – DATETIME</td>
<td>INTERVAL</td>
</tr>
<tr>
<td>DATETIME – DATE</td>
<td>INTERVAL</td>
</tr>
<tr>
<td>DATE + or – INTERVAL</td>
<td>DATETIME</td>
</tr>
</tbody>
</table>

In the cases that Table 2-8 shows, DATE values are first converted to their corresponding DATETIME equivalents, and then the expression is evaluated by the rules of arithmetic.

Although you can interchange DATE and DATETIME values in many situations, you must indicate whether a value is a DATE or a DATETIME data type. A DATE value can come from the following sources:

- A column or program variable of type DATE
- The TODAY keyword
- The DATE() function
- The MDY function
- A DATE literal

A DATETIME value can come from the following sources:

- A column or program variable of type DATETIME
- The CURRENT keyword
- The EXTEND function
- A DATETIME literal

The database locale defines the default DATE and DATETIME formats. For the default locale, U.S. English, these formats are ‘mm/dd/yy’ for DATE values and ‘yyyy-mm-dd hh:MM:ss’ for DATETIME values.

To represent DATE and DATETIME values as character strings, the fields in the strings must be in proper order. In other words, when a DATE value is expected, the string must be in DATE format and when a DATETIME value is expected, the string must be in DATETIME format. For example, you can use the string 10/30/2008 as a DATE string but not as a DATETIME string. Instead, you must use 2008-10-30 or 08-10-30 as the DATETIME string.

In a nondefault locale, literal DATE and DATETIME strings must match the formats that the locale defines. For more information, see the IBM Informix GLS User’s Guide.

You can customize the DATE format that the database server expects with the DBDATE and GL_DATE environment variables. You can customize the DATETIME format that the database server expects with the DBTIME and GL_DATETIME environment variables. For more information, see “DBDATE” on page 3-20 and “DBTIME” on page 3-31. For more information on all these environment variables, see the IBM Informix GLS User’s Guide.

You can also subtract one DATE value from another DATE value, but the result is a positive or negative INTEGER count of days, rather than an INTERVAL value. If
an INTERVAL value is required, you can either use the UNITS DAY operator to convert the INTEGER value into an INTERVAL DAY TO DAY value, or else use EXTEND to convert one of the DATE values into a DATETIME value before subtracting.

For example, the following expression uses the DATE( ) function to convert character string constants to DATE values, calculates their difference, and then uses the UNITS DAY keywords to convert the INTEGER result into an INTERVAL value:

\[
\text{DATE ('5/2/2007') - DATE ('4/6/1968')} \text{ UNITS DAY}
\]

Result: INTERVAL (12810) DAY(5) TO DAY

**Important:** Because of the high precedence of UNITS relative to other SQL operators, you should generally enclose any arithmetic expression that is the operand of UNITS within parentheses, as in the preceding example.

If you need YEAR TO MONTH precision, you can use the EXTEND function on the first DATE operand, as the following example shows:

\[
\text{EXTEND (DATE ('5/2/2007'), YEAR TO MONTH) - DATE ('4/6/1969')}
\]

Result: INTERVAL (39-01) YEAR TO MONTH

The resulting INTERVAL precision is YEAR TO MONTH, because the DATETIME value came first. If the DATE value had come first, the resulting INTERVAL precision would have been DAY(5) TO DAY.

**Manipulating INTERVAL Values**

You can add or subtract INTERVAL values only if both values are from the same class; that is, if both are year-month or both are day-time. In the following example, a SECOND TO FRACTION value is subtracted from a MINUTE TO FRACTION value:

\[
\text{INTERVAL (100:30.0005) MINUTE(3) TO FRACTION(4) - INTERVAL (120.01) SECOND(3) TO FRACTION}
\]

Result: INTERVAL (98:29.9905) MINUTE TO FRACTION(4)

The use of numeric qualifiers alerts the database server that the MINUTE and FRACTION in the first value and the SECOND in the second value exceed the default number of digits.

When you add or subtract INTERVAL values, the second value cannot have a field with greater precision than the first. The second INTERVAL, however, can have a field of smaller precision than the first. For example, the second INTERVAL can be HOUR TO SECOND when the first is DAY TO HOUR. The additional fields (in this case MINUTE and SECOND) in the second INTERVAL value are ignored in the calculation.

**Multiplying or Dividing INTERVAL Values**

You can multiply or divide INTERVAL values by numbers. Any remainder from the calculation is ignored, however, and the result is truncated to the precision of the INTERVAL. The following expression multiplies an INTERVAL value by a literal number that has a fractional part:

\[
\text{INTERVAL (15:30.0002) MINUTE TO FRACTION(4) * 2.5}
\]

Result: INTERVAL (38:45.0005) MINUTE TO FRACTION(4)
In this example, 15 * 2.5 = 37.5 minutes, 30 * 2.5 = 75 seconds, and 2 * 2.5 = 5 FRACTION (4). The 0.5 minute is converted into 30 seconds and 60 seconds are converted into 1 minute, which produces the final result of 38 minutes, 45 seconds, and 0.0005 of a second. The result of any calculation has the same precision as the original INTERVAL operand.

Extended Data Types (IDS)

Dynamic Server enables you to create extended data types to characterize data that cannot easily be represented with the built-in data types. (You cannot, however, use extended data types in distributed transactions that query external tables.) You can create these categories of extended data types:

- Complex data types
- Distinct data types
- Opaque data types

Sections that follow provide an overview of each of these data types.

For more information about extended data types, see the IBM Informix Database Design and Implementation Guide and IBM Informix User-Defined Routines and Data Types Developer’s Guide.

Complex Data Types

A complex data type can store one or more values of other built-in and extended data types. Figure 2-5 shows the complex types that Dynamic Server supports.

Figure 2-5. Complex Data Types of Dynamic Server

The following table summarizes the structure of the complex data types.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection types:</td>
<td>Complex data types that are made up of elements, each of which is of the same data type.</td>
</tr>
<tr>
<td>LIST</td>
<td>A group of ordered elements, each of which need not be unique within the group.</td>
</tr>
<tr>
<td>MULTISET</td>
<td>A group of elements, each of which need not be unique. The order of the elements is ignored.</td>
</tr>
<tr>
<td>SET</td>
<td>A group of elements, each of which is unique. The order of the elements is ignored.</td>
</tr>
<tr>
<td>ROW types:</td>
<td>Complex data types that are made up of fields.</td>
</tr>
<tr>
<td>Named ROW type</td>
<td>Row types that are identified by their name.</td>
</tr>
<tr>
<td>Unnamed ROW type</td>
<td>Row types that are identified by their structure.</td>
</tr>
</tbody>
</table>
Complex data types can be nested. For example, you can construct a ROW type whose fields include one or more sets, multisets, ROW types, and lists. Likewise, a collection type can have elements whose data type is a ROW type or a collection type.

Complex types that include opaque types inherit the following support functions.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>export</td>
<td>LO_handles</td>
</tr>
<tr>
<td>output</td>
<td>import_binary</td>
<td>hash</td>
</tr>
<tr>
<td>send</td>
<td>export_binary</td>
<td>lessthan</td>
</tr>
<tr>
<td>recv</td>
<td>assign</td>
<td>equal</td>
</tr>
<tr>
<td>import</td>
<td>destroy</td>
<td>lessthan (ROW only)</td>
</tr>
</tbody>
</table>

Sections that follow summarize the complex data types. For more information, see the IBM Informix Database Design and Implementation Guide.

**Collection Data Types**

A collection data type is a complex type that is made up of one or more elements, all of the same data type. A collection element can be of any data type (including other complex types) except BYTE, TEXT, SERIAL, or SERIAL8.

**Important:** An element cannot have a *NULL* value. You must specify the *NOT NULL* constraint for collection elements. No other constraints are valid for collections.

Dynamic Server supports three kinds of built-in collection types: LIST, SET, and MULTISET. The keywords used to declare these collections are the names of the *type constructors* or just *constructors*. For the syntax of collection types, see the IBM Informix Guide to SQL: Syntax. No more than 97 columns of the same table can be declared as collection data types.

When you specify element values for a collection, list the element values after the constructor and between braces ({}). For example, suppose you have a collection column with the following MULTISET data type:

```
CREATE TABLE table1
(
    mset_col MULTISET(INTEGER NOT NULL)
)
```

The next INSERT statement adds one group of element values to this column. (The word MULTISET in these two examples is the MULTISET constructor.)

```
INSERT INTO table1 VALUES (MULTISET(5, 9, 7, 5))
```

You can leave the braces empty to indicate an empty set:

```
INSERT INTO table1 VALUE (MULTISET{})
```

An empty collection is not equivalent to a NULL value for the column.

**Accessing Collection Data:** To access the elements of a collection column, you must fetch the collection into a collection variable and modify the contents of the collection variable. Collection variables can be either of the following types:

- Variables in an SPL routine
  
  For more information, see the IBM Informix Guide to SQL: Tutorial.
• Host variables in an IBM Informix ESQL/C program
  For more information, see the IBM Informix ESQL/C Programmer’s Manual.

You can also use nested dot notation to access collection data. For more about accessing elements of a collection, see the IBM Informix Guide to SQL: Tutorial.

**Important:** Collection data types are not valid as arguments to functions that are used for functional indexes.

**ROW Data Types**

A ROW data type is an ordered collection of one or more elements, called fields. Each field has a name and a data type. The fields of a ROW are comparable to the columns of a table, but with important differences:

• A field has no default clause.
• You cannot define constraints on a field.
• You can only use fields with row types, not with tables.

Two kinds of ROW data types exist:

• *Named ROW data types* are identified by their names.
• *Unnamed ROW data types* are identified by their structure.

The structure of an unnamed ROW data type is the number (and the order of data types) of its fields.

No more than 195 columns of the same table can be declared as ROW data types. For more information about ROW data types, see “ROW, Named (IDS)” on page 2-23 and “ROW, Unnamed (IDS)” on page 2-24.

You can cast between named and unnamed ROW data types; this is described in the IBM Informix Database Design and Implementation Guide.

**Distinct Data Types**

A distinct data type has the same internal structure as some other source data type in the database. The source type can be a built-in or extended data type. What distinguishes a distinct type from its source type are support functions that are defined on the distinct type.

No more than 195 columns in the same table can be DISTINCT types that are based on BYTE, TEXT, ROW, VARCHAR, NVARCHAR, or VARCHAR source types, and no more than 97 can have collection source types. For more information, see the section “Distinct (IDS)” on page 2-14. See also IBM Informix User-Defined Routines and Data Types Developer’s Guide.

**Opaque Data Types**

An opaque data type is a user-defined data type that is fully encapsulated. That is, its internal structure is unknown to the database server. User-defined types (UDTs) that are not DISTINCT types whose source types are built-in types are opaque.

The built-in data types BLOB, BOOLEAN, CLOB, and VARCHAR are implemented as opaque data types. You cannot access these built-in opaque data types in cross-server distributed operations, but you can access them in other databases of the same Dynamic Server instance.
Data Type Casting and Conversion

Occasionally, the data type that was assigned to a column with the CREATE TABLE statement is inappropriate. You might wish to change the data type of a column when you need to store larger values than the current data type can accommodate. The database server allows you to change the data type of the column or to cast its values to a different data type with either of the following methods:

- Use the ALTER TABLE statement to modify the data type of a column.
  For example, if you create a SMALLINT column and later find that you need to store integers larger than 32,767, you must change the data type of that column to store the larger value. You can use ALTER TABLE to change the data type to INTEGER. The conversion changes the data type of all values that currently exist in the column as well as any new values that might be added.
- Use the CAST AS keywords or the double colon (::) cast operator to cast a value to a different data type.
  Casting does not permanently alter the data type of a value; it expresses the value in a more convenient form. Casting user-defined data types into built-in types allows client programs to manipulate data types without knowledge of their internal structure.

If you change data types, the new data type must be able to store all of the old value.

Both data-type conversion and casting depend on casts registered in the syscasts system catalog table. For information about syscasts, see “SYSCASTS (IDS)” on page 1-14.

A cast is either built-in or user defined. Guidelines exist for casting distinct and extended data types. For more information about casting opaque data types, see IBM Informix User-Defined Routines and Data Types Developer’s Guide. For information about casting other extended data types see, the IBM Informix Database Design and Implementation Guide.

Using Built-in Casts

User informix owns built-in casts. They govern conversions from one built-in data type to another. Built-in casts allow the database server to attempt the following data-type conversions:

- A character type to any other character type
- A character type to or from another built-in type
- A numeric type to any other numeric type

The database server automatically invokes appropriate built-in casts when required. For time data types, conversion between DATE and DATETIME data types requires explicit casts with the EXTEND function, and explicit casts with the UNITS operator are required for number-to-INTERVAL conversion. Built-in casts are not available for converting large (BYTE, BLOB, CLOB, and TEXT) built-in types to other built-in data types.
When you convert a column from one built-in data type to another, the database server applies the appropriate built-in casts to each value already in the column. If the new data type cannot store any of the resulting values, the ALTER TABLE statement fails.

For example, if you try to convert a column from the INTEGER data type to the SMALLINT data type and the following values exist in the INTEGER column, the database server does not change the data type, because SMALLINT columns cannot accommodate numbers greater than 32,767:

\[
100 \\ 400 \\ 700 \\ 50000 \\ 700
\]

The same situation might occur if you attempt to transfer data from FLOAT or SMALLFLOAT columns to INTEGER, SMALLINT, or DECIMAL columns. Errors of overflow, underflow, or truncation can occur during data type conversion.

Sections that follow describe database server behavior during certain types of casts and conversions.

**Converting from Number to Number**

When you convert data from one number data type to another, you occasionally find rounding errors. The following table indicates which numeric data type conversions are acceptable and what kinds of errors you can encounter when you convert between certain numeric data types.

<table>
<thead>
<tr>
<th>Target Type</th>
<th>SMALL INT</th>
<th>INTEGER</th>
<th>INT8</th>
<th>SMALL FLOAT</th>
<th>FLOAT</th>
<th>DECIMAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALLINT</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>INTEGER</td>
<td>E</td>
<td>OK</td>
<td>OK</td>
<td>E</td>
<td>OK</td>
<td>P</td>
</tr>
<tr>
<td>INT8</td>
<td>E</td>
<td>E</td>
<td>OK</td>
<td>D</td>
<td>E</td>
<td>P</td>
</tr>
<tr>
<td>SMALLFLOAT</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>OK</td>
<td>OK</td>
<td>P</td>
</tr>
<tr>
<td>FLOAT</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>D</td>
<td>OK</td>
<td>P</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>E</td>
<td>E</td>
<td>E</td>
<td>D</td>
<td>D</td>
<td>P</td>
</tr>
</tbody>
</table>

Legend:

- **OK**: No error
- **P**: An error can occur, depending on the precision of the decimal
- **E**: An error can occur, depending on the data value
- **D**: No error, but less significant digits might be lost

For example, if you convert a FLOAT value to DECIMAL(4,2), your database server rounds off the floating-point number before storing it as DECIMAL.

This conversion can result in an error depending on the precision assigned to the DECIMAL column.

**Converting Between Number and Character**

You can convert a character column (of a data type such as CHAR, NCHAR, NVARCHAR, or VARCHAR) to a numeric column. If a data string, however, contains any characters that are not valid in a number column (for example, the letter l instead of the number 1), the database server returns an error.
You can also convert a numeric column to a character column. If the character column is not large enough to receive the number, however, the database server generates an error. If the database server generates an error, it cannot complete the ALTER TABLE statement or cast, and leaves the column values as characters. You receive an error message and the statement is rolled back automatically (regardless of whether you are in a transaction).

**Converting Between INTEGER and DATE**
You can convert an integer column (SMALLINT, INTEGER, or INT8) to a DATE value. The database server interprets the integer as a value in the internal format of the DATE column. You can also convert a DATE column to an integer column. The database server stores the internal format of the DATE column as an integer representing a Julian date.

**Converting Between DATE and DATETIME**
You can convert DATE columns to DATETIME columns. If the DATETIME column contains more fields than the DATE column, however, the database server either ignores the fields or fills them with zeros. The illustrations in the following list show how these two data types are converted (assuming that the default date format is mm/dd/yyyy):

- If you convert DATE to DATETIME YEAR TO DAY, the database server converts the existing DATE values to DATETIME values. For example, the value 08/15/2002 becomes 2002-08-15.
- If you convert DATETIME YEAR TO DAY to the DATE format, the value 2002-08-15 becomes 08/15/2002.
- If you convert DATE to DATETIME YEAR TO SECOND, the database server converts existing DATE values to DATETIME values and fills in the additional DATETIME fields with zeros. For example, 08/15/2002 becomes 2002-08-15 00:00:00.
- If you convert DATETIME YEAR TO SECOND to DATE, the database server converts existing DATETIME to DATE values but drops fields for time units smaller than DAY. For example, 2002-08-15 12:15:37 becomes 08/15/2002.

**Using User-Defined Casts**
Implicit and explicit casts are owned by the users who create them. They govern casts and conversions between user-defined data types and other data types. Developers of user-defined data types must create certain implicit and explicit casts and the functions that are used to implement them. The casts allow user-defined types to be expressed in a form that clients can manipulate.

For information on how to register and use implicit and explicit casts, see the CREATE CAST statement in the *IBM Informix Guide to SQL: Syntax* and the *IBM Informix Database Design and Implementation Guide.*

**Implicit Casts**
Implicit casts allow you to convert a user-defined data type to a built-in type or vice versa. The database server automatically invokes a single implicit cast when needed to evaluate and compare expressions or pass arguments. Operations that require more than one implicit cast fail.

Users can explicitly invoke an implicit cast using the CAST AS keywords or the double colon (: ) cast operator.
Explicit Casts
Explicit casts, unlike implicit casts or built-in casts, are never invoked automatically by the database server. Users must invoke them explicitly with the CAST AS keywords or with the double colon (::) cast operator.

Determining Which Cast to Apply
The database server uses the following rules to determine which cast to apply in a particular situation:

- To compare two built-in types, the database server automatically invokes the appropriate built-in casts.
- The database server applies only one implicit cast per operand. If two or more casts are needed to convert the operand to the desired type, the user must explicitly invoke the additional casts.

In the following example, the literal value 5.55 is implicitly cast to DECIMAL, and is then explicitly cast to MONEY, and finally to yen:

```
CREATE DISTINCT TYPE yen AS MONEY

INSERT INTO currency_tab
VALUES (5.55::MONEY::yen)
```

- To compare a distinct type to its source type, the user must explicitly cast one type to the other.
- To compare a distinct type to a type other than its source, the database server looks for an implicit cast between the source type and the desired type.

If neither cast is registered, the user must invoke an explicit cast between the distinct type and the desired type. If this cast is not registered, the database server automatically invokes a cast from the source type to the desired type.

If none of these casts is defined, the comparison fails.

- To compare an opaque type to a built-in type, the user must explicitly cast the opaque type to a data type that the database server understands (such as LVARCHAR, SENDRECV, IMPEX, or IMPEXBIN). The database server then invokes built-in casts to convert the results to the desired built-in type.
- To compare two opaque types, the user must explicitly cast one opaque type to a form that the database server understands (such as LVARCHAR, SENDRECV, IMPEX, or IMPEXBIN) and then explicitly cast this type to the second opaque type.

For information about casting and the IMPEX, IMPEXBIN, LVARCHAR, and SENDRECV types, see IBM Informix User-Defined Routines and Data Types Developer's Guide.

Casts for Distinct Types
You define a distinct type based on a built-in type or an existing opaque type or ROW type. Although data of the distinct type has the same length and alignment and is passed in the same way as data of the source type, the two cannot be compared directly. To compare a distinct type and its source type, you must explicitly cast one type to the other.

When you create a new distinct type, the database server automatically registers two explicit casts:

- A cast from the distinct type to its source type
- A cast from the source type to the distinct type
You can create an implicit cast between a distinct type and its source type. To create an implicit cast, however, you must first drop the default explicit cast between the distinct type and its source type.

You also can use all casts that have been registered for the source type without modification on the distinct type. You can also create and register new casts and support functions that apply only to the distinct type.

For examples that show how to create a cast function for a distinct type and register the function as cast, see the IBM Informix Database Design and Implementation Guide.

**Important:** For releases of Dynamic Server earlier than Version 9.21, distinct data types inherited the built-in casts that are provided for the source type. The built-in casts of the source type are not inherited by distinct data types in this release.

### What Extended Data Types Can Be Cast?

The next table shows the extended data type combinations that you can cast.

<table>
<thead>
<tr>
<th>Target Type</th>
<th>Opaque Type</th>
<th>Distinct Type</th>
<th>Named ROW Type</th>
<th>Unnamed ROW Type</th>
<th>Collection Type</th>
<th>Built-in Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Opaque Type</td>
<td>Explicit or implicit</td>
<td>Explicit</td>
<td>Explicit</td>
<td>Not Valid</td>
<td>Not Valid</td>
<td>Explicit or implicit</td>
</tr>
<tr>
<td>Distinct Type</td>
<td>Explicit</td>
<td>Explicit</td>
<td>Explicit</td>
<td>Not Valid</td>
<td>Not Valid</td>
<td>Explicit or implicit</td>
</tr>
<tr>
<td>Named ROW Type</td>
<td>Explicit</td>
<td>Explicit</td>
<td>Explicit</td>
<td>Not Valid</td>
<td>Not Valid</td>
<td>Not Valid</td>
</tr>
<tr>
<td>Unnamed ROW Type</td>
<td>Not Valid</td>
<td>Not Valid</td>
<td>Explicit</td>
<td>Implicit</td>
<td>Not Valid</td>
<td>Not Valid</td>
</tr>
<tr>
<td>Collection Type</td>
<td>Not Valid</td>
<td>Not Valid</td>
<td>Not Valid</td>
<td>Not Valid</td>
<td>Explicit</td>
<td>Not Valid</td>
</tr>
<tr>
<td>Built-in Type</td>
<td>Explicit or implicit</td>
<td>Explicit</td>
<td>Not Valid</td>
<td>Not Valid</td>
<td>Not Valid</td>
<td>System defined (implicit)</td>
</tr>
</tbody>
</table>

1 Applies when two ROW types are structurally equivalent or casts exist to handle data conversions where corresponding field types are not the same. 2 Applies when a cast exists to convert between the element types of the respective collection types. 3 Applies when a user-defined cast exists to convert between the two data types.

The table shows only whether or not a cast between a source type and a target type are possible. In some cases, you must first create a user-defined cast before you can perform a conversion between two data types. In other cases, the database server provides either an implicit cast or a built-in cast that you must explicitly invoke.

### Operator Precedence

An operator is a symbol or keyword that can appear in an SQL expression. Most SQL operators are restricted in the data types of their operands and returned values. Some operators only support operands of built-in data types; others can support built-in and extended data types as operands.
The following table shows the precedence of the operators that Informix database servers support, in descending (highest to lowest) order of precedence. Operators with the same precedence are listed in the same row.

<table>
<thead>
<tr>
<th>Operator Precedence</th>
<th>Example in Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>. (membership)</td>
<td>customer.phone [1, 3]</td>
</tr>
<tr>
<td>[ ] (substring)</td>
<td></td>
</tr>
<tr>
<td>UNITS</td>
<td>x UNITS DAY</td>
</tr>
<tr>
<td>+</td>
<td></td>
</tr>
<tr>
<td>- (unary)</td>
<td>- y</td>
</tr>
<tr>
<td>:: (cast)</td>
<td>NULL::TEXT</td>
</tr>
<tr>
<td>*</td>
<td>x / y</td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>x - y</td>
</tr>
<tr>
<td>- (binary)</td>
<td></td>
</tr>
<tr>
<td>l l (concatenation)</td>
<td>customer.fname l l customer.lname</td>
</tr>
<tr>
<td>ANY</td>
<td>orders.ship_date &gt; SOME</td>
</tr>
<tr>
<td>ALL</td>
<td>(SELECT paid_date FROM orders)</td>
</tr>
<tr>
<td>SOME</td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td>NOT y</td>
</tr>
<tr>
<td>&lt;</td>
<td>x &gt;= y</td>
</tr>
<tr>
<td>&lt;=</td>
<td></td>
</tr>
<tr>
<td>=</td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td></td>
</tr>
<tr>
<td>&gt;=</td>
<td></td>
</tr>
<tr>
<td>!=</td>
<td></td>
</tr>
<tr>
<td>&lt;&gt;</td>
<td></td>
</tr>
<tr>
<td>IN</td>
<td></td>
</tr>
<tr>
<td>BETWEEN ... AND</td>
<td>customer.fname MATCHES y</td>
</tr>
<tr>
<td>LIKE</td>
<td></td>
</tr>
<tr>
<td>MATCHES</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td>x AND y</td>
</tr>
<tr>
<td>OR</td>
<td>x OR y</td>
</tr>
</tbody>
</table>

See the *IBM Informix Guide to SQL: Syntax* for the syntax and semantics of these SQL operators.
Chapter 3. Environment Variables

In This Chapter .......................... 3-3
Types of Environment Variables .......... 3-3
Using Environment Variables on UNIX . 3-3
  Where to Set Environment Variables on UNIX .......... 3-4
  Setting Environment Variables in a Configuration File . 3-4
  Setting Environment Variables at Login Time . 3-5
Syntax for Setting Environment Variables . 3-5
Unsetting Environment Variables . 3-6
Modifying an Environment-Variable Setting . 3-6
Viewing Your Environment-Variable Settings . 3-6
Checking Environment Variables with the chkv1 utility . 3-6
Rules of Precedence .......................... 3-7
Using Environment Variables on Windows . 3-8
  Where to Set Environment Variables on Windows . 3-8
Environment Settings . 3-8
  Using the System Applet to Change Environment Variables . 3-8
  Using the Command Prompt to Change Environment Variables . 3-9
  Using dbservername.cmd to Initialize a Command-Prompt Environment . 3-9
Rules of Precedence ......................... 3-10
List of Environment Variables .............. 3-10
Environment Variables ................. 3-14
  AC_CONFIG ................................ 3-14
  AFDEBUG ................................ 3-14
  ANSIOWNER (IDS) .......................... 3-14
  BIG_FET_BUF_SIZE (XPS) .................. 3-15
  CPFIRST ................................ 3-15
  DBACCMOIGN .............................. 3-16
  DBANSWARN ................................ 3-17
  DBBLOBBUF ................................ 3-17
  DBCENTURY ................................ 3-18
  DBDATE ................................ 3-20
  DBDELETRMTER ............................ 3-22
  DBEDIT ................................ 3-22
  DBFMTMASK ................................ 3-23
  DBLANG ................................ 3-23
  DBMONEY ................................ 3-24
  DBNLS (IDS) .............................. 3-25
  DBONLOAD (IDS) ........................... 3-26
  DBPATH ................................ 3-26
  DBPRINT ................................ 3-28
  DBREMOTECMD (UNIX) ...................... 3-28
  DBSPACETEMP ............................. 3-29
  DBTEMP (IDS) ............................. 3-30
  DBTIME ................................ 3-31
  DBUPSPACE ................................ 3-33
  DEFAULT_ATTACH ................................ 3-34
  DELIMIDENT .............................. 3-34
  ENVIGNORE (UNIX) ......................... 3-35
  FET_BUF_SIZE ............................. 3-36
  GLOBAL_DETACH_INFORM (XPS) ............... 3-36
  IBM_XPS_PARAMS (XPS) .................... 3-37
  IFM_CART_ALRM (XPS) ..................... 3-38
  IFM_HISTORY_SIZE (XPS) .................. 3-38
  IFM_OPT_FACT_TABS (XPS) ................. 3-38
  IFM_OPT_NON_DIA_TABS (XPS) ............... 3-39
IFX_DEF_TABLE_LOCKMODE (IDS) ........................................ 3-39
IFX_DIRECTIVES ................................................... 3-40
IFX_EXTDIRECTIVES ................................................ 3-41
IFX_LONGID ....................................................... 3-41
IFX_NETBUF_PVTPOOL_SIZE (UNIX) ............................... 3-42
IFX_NETBUF_SIZE ................................................ 3-42
IFX_NO_LRU_TUNING .............................................. 3-43
IFX_NO_TIMELIMIT_WARNING ..................................... 3-43
IFX_NODBPROC ................................................... 3-43
IFX_ONTAPE_FILE_PREFIX ....................................... 3-44
IFX_PAD_VARCHAR (IDS) ......................................... 3-44
IFX_UPDDESC (IDS) ................................................. 3-44
IFX_XASTCOMPLIANCE_XAEND ..................................... 3-45
IFX_XFER_SHMBASE .............................................. 3-45
IMADMIN .......................................................... 3-45
IMC_CONFIG ...................................................... 3-46
IMCSERVER ....................................................... 3-46
INFORMIXC (UNIX) ................................................ 3-47
INFORMIXCONCSMCFG (IDS) ...................................... 3-47
INFORMIXCONRETRY ............................................. 3-47
INFORMIXCONTIME ............................................... 3-48
INFORMIXCPPMAP (IDS) ......................................... 3-49
INFORMIXDIR ..................................................... 3-49
INFORMIXKEYTAB (UNIX) ........................................ 3-49
INFORMIXPCACHE (IDS) .......................................... 3-50
INFORMIXSERVER ................................................. 3-50
INFORMIXSHMBASE (UNIX) ....................................... 3-51
INFORMIXSQLHOSTS ............................................. 3-51
INFORMIXSTACKSIZE ............................................ 3-52
INFORMIXTERM (UNIX) ........................................... 3-52
INF_ROLE_SEP (IDS) .............................................. 3-53
INTERACTIVE_DESKTOP_OFF (Windows) ......................... 3-54
ISM_COMPRESSION ................................................. 3-54
ISM_DEBUG_FILE .................................................. 3-54
ISM_DEBUG_LEVEL ............................................... 3-55
ISM_ENCRYPTION ............................................... 3-55
ISM_MAXLOGSIZE ................................................ 3-55
ISM_MAXLOGVERS ................................................. 3-55
JAR_TEMP_PATH (IDS) ............................................ 3-56
JAVA_COMPILER (IDS) .......................................... 3-56
JVM_MAX_HEAP_SIZE (IDS) ....................................... 3-56
LD_LIBRARY_PATH (UNIX) ........................................ 3-57
LIBRARYMATCH (XPS) ........................................... 3-57
LIBPATH (UNIX) .................................................. 3-57
NODEFAC (UNIX) .................................................. 3-58
ONCONFIG ........................................................ 3-58
OPTCOMPIND ..................................................... 3-59
OPTMSG .......................................................... 3-59
OPTOFC .......................................................... 3-60
OPT_GOAL (IDS, UNIX) .......................................... 3-60
PATH .............................................................. 3-61
PDQ_PRIORITY .................................................... 3-61
    Using PDQ_PRIORITY with Dynamic Server ................ 3-62
    Using PDQ_PRIORITY with Extended Parallel Server .... 3-62
PLCONFIG (IDS) .................................................. 3-62
PLOAD_LO_PATH (IDS) .......................................... 3-63
PLOAD_SHMBASE (IDS) .......................................... 3-63
PSORT_DBTEMP .................................................. 3-63
PSORT_NPROCS .................................................. 3-64
RTREE_COST_ADJUST_VALUE (IDS) .............................. 3-65
SHLIB_PATH (UNIX) ............................................. 3-65
In This Chapter

Various environment variables affect the functionality of your IBM Informix products. You can set environment variables that identify your terminal, specify the location of your software and define other parameters.

Some environment variables are required; others are optional. You must either set or accept the default setting for required environment variables.

This chapter describes how to use the environment variables that apply to one or more IBM Informix products and shows how to set them.

Types of Environment Variables

Two types of environment variables are discussed in this chapter:

- Informix-specific environment variables
  Set Informix environment variables when you want to work with IBM Informix products. Each IBM Informix product publication specifies the environment variables that you must set to use that product.

- Operating-system-specific environment variables
  IBM Informix products rely on the correct setting of certain standard operating system environment variables. For example, you must always set the PATH environment variable.

In a UNIX environment, you might also need to set the TERMINFO or TERM Environment variable to use some products effectively.

The GLS environment variables that support nondefault locales are described in the IBM Informix GLS User’s Guide. The GLS variables are included in the list of environment variables in Table 3-1 on page 3-10 and in the topic index in Table 3-4 on page 3-69, but are not discussed in this publication.

The database server uses the environment variables that were in effect at the time when the database server was initialized.

The onstat -g env command lists the active environment settings.

Tip: Additional environment variables that are specific to your client application or SQL API might be discussed in the publication for that product.

Important: Do not set any environment variable in the home directory of user informix (or in the file .informix in that directory) while initializing the database and creating the sysmaster database."
Using Environment Variables on UNIX

The following sections discuss setting unseting modifying and viewing environment variables. If you already use an IBM Informix product some or all of the appropriate environment variables might be set.

Where to Set Environment Variables on UNIX

You can set environment variables on UNIX in the following places:

- At the system prompt on the command line
  When you set an environment variable at the system prompt, you must reassign it the next time you log into the system. See also "Using Environment Variables on UNIX" on page 3-4

- In an environment-configuration file
  An environment-configuration file is a common or private file where you can set all the environment variables that IBM Informix products use. The use of such files reduces the number of environment variables that you must set at the command line or in a shell file.

- In a login file
  Values of environment variables set in your .login, .cshrc, or .profile file are assigned automatically every time you log into the system.

- In the SET ENVIRONMENT statement of SQL
  Values of some environment variables can reset by the SET ENVIRONMENT statement. The scope of the new settings is generally the routine that executed the SET ENVIRONMENT statement, but it is the current session for the OPTCOMPIND environment variable of Dynamic Server, as described in the section OPTCOMPIND environment variable of Dynamic Server, as described in the section "OPTCOMPIND" on page 3-59 and for environment variables of Extended Parallel Server that the sysdbopen() or sysdbclose() SPL routines can set. For more information on these routines and on the SET ENVIRONMENT statement, see the IBM Informix Guide to SQL: Syntax.

In IBM Informix ESQL/C, you can set supported environment variables within an application with the putenv() system call and retrieve values with the getenv() system call, if your UNIX system supports these functions. For more information on putenv() and getenv(), see the IBM Informix ESQL/C Programmer’s Manual and your C documentation.

Setting Environment Variables in a Configuration File

The common (shared) environment-configuration file that is provided with IBM Informix products resides in $INFORMIXDIR/etc/informix.rc. Permissions for this shared file must be set to 644.

A user can override the system or shared environment variables by setting variables in a private environment-configuration file. This file must have all of the following characteristics:

- Stored in the user’s home directory
- Named .informix
- Permissions set to readable by the user

An environment-configuration file can contain comment lines (preceded by the # comment indicator) and variable definition lines that set values (separated by blank spaces or tabs), as the following example shows:
You can use the ENVIGNORE environment variable, described in “ENVIGNORE (UNIX)” on page 3-35, to override one or more entries in an environment-configuration file. Use the Informix chkenv utility, described in “Checking Environment Variables with the chkenv Utility” on page 3-6, to perform a sanity check on the contents of an environment-configuration file. The chkenv utility returns an error message if the file contains a bad environment variable or if the file is too large.

The first time you set an environment variable in a shell file or environment-configuration file, you must tell the shell process to read your entry before you work with your IBM Informix product. If you use a C shell, source the file; if you use a Bourne or Korn shell, use a period (.) to execute the file.

**Setting Environment Variables at Login Time**

Add commands that set your environment variables to the appropriate login file:

For C shell

```
.login or .cshrc
```

For Bourne shell or Korn shell

```
.profile
```

**Syntax for Setting Environment Variables**

Use standard UNIX commands to set environment variables. The examples in the following table show how to set the ABCD environment variable to value for the C shell, Bourne shell, and Korn shell. The Korn shell also supports a shortcut, as the last row indicates. Environment variables are case sensitive.

<table>
<thead>
<tr>
<th>Shell</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td><code>setenv ABCD value</code></td>
</tr>
<tr>
<td>Bourne</td>
<td><code>ABCD=value</code></td>
</tr>
<tr>
<td></td>
<td><code>export ABCD</code></td>
</tr>
<tr>
<td>Korn</td>
<td><code>ABCD=value</code></td>
</tr>
<tr>
<td></td>
<td><code>export ABCD</code></td>
</tr>
<tr>
<td>Korn</td>
<td><code>export ABCD=value</code></td>
</tr>
</tbody>
</table>

The following diagram shows how the syntax for setting an environment variable is represented throughout this chapter. These diagrams indicate the setting for the C shell; for the Bourne or Korn shells, use the syntax illustrated in the preceding table.

```
>>>setenv--ABCD=value
```

For more information on how to read syntax diagrams, see “Syntax Diagrams” on page xv of the Introduction.
Unsetting Environment Variables

To unset an environment variable, enter the following command.

<table>
<thead>
<tr>
<th>Shell</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>unsetenv ABCD</td>
</tr>
<tr>
<td>Bourne or Korn</td>
<td>unset ABCD</td>
</tr>
</tbody>
</table>

Modifying an Environment-Variable Setting

Sometimes you must add information to an environment variable that is already set. For example, the PATH environment variable is always set on UNIX. When you use an IBM Informix product, you must add to the PATH setting the name of the directory where the executable files for the IBM Informix products are stored.

In the following example, the INFORMIXDIR is /usr/informix. (That is, during installation, the IBM Informix products were installed in the /usr/informix directory.) The executable files are in the bin subdirectory, /usr/informix/bin. To add this directory to the front of the C shell PATH environment variable, use the following command:

```
setenv PATH /usr/informix/bin:$PATH
```

Rather than entering an explicit pathname, you can use the value of the INFORMIXDIR environment variable (represented as $INFORMIXDIR), as the following example shows:

```
setenv INFORMIXDIR /usr/informix
setenv PATH $INFORMIXDIR/bin:$PATH
```

You might prefer to use this version to ensure that your PATH entry does not conflict with the search path that was set in INFORMIXDIR, and so that you do not have to reset PATH whenever you change INFORMIXDIR. If you set the PATH environment variable on the C shell command line, you might need to include braces ({} ) with the existing INFORMIXDIR and PATH, as the following command shows:

```
setenv PATH ${INFORMIXDIR}/bin:${PATH}
```

For more information about how to set and modify environment variables, refer to the publications for your operating system.

Viewing Your Environment-Variable Settings

After you install one or more IBM Informix products, enter the following command at the system prompt to view your current environment settings.

<table>
<thead>
<tr>
<th>UNIX Version</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>BSD UNIX</td>
<td>env</td>
</tr>
<tr>
<td>UNIX System V</td>
<td>printenv</td>
</tr>
</tbody>
</table>

Checking Environment Variables with the chkenv Utility

The chkenv utility checks the validity of shared or private environment-configuration files. It validates the names of the environment variables in the file, but not their values. Use chkenv to provide debugging information when you define, in an environment-configuration file, all the environment variables that your IBM Informix products use.
filename is the name of the environment-configuration file to be debugged.
pathname is the full directory path in which the environment variable file is located.

File $INFORMIXDIR/etc/informix.rc is the shared environment-configuration file. A private environment-configuration file is stored as .informix in the home directory of the user. If you specify no pathname for chkenv, the utility checks both the shared and private environment configuration files. If you provide a pathname, chkenv checks only the specified file.

Issue the following command to check the contents of the shared environment-configuration file:

```
chkenv informix.rc
```

The chkenv utility returns an error message if it finds a bad environment-variable name in the file or if the file is too large. You can modify the file and rerun the utility to check the modified environment-variable names.

IBM Informix products ignore all lines in the environment-configuration file, starting at the point of the error, if the chkenv utility returns the following message:

```
-33523 filename: Bad environment variable on line number.
```

If you want the product to ignore specified environment-variables in the file, you can also set the ENVIGNORE environment variable. For a discussion of the use and format of environment-configuration files and the ENVIGNORE environment variable, see page 3-35.

### Rules of Precedence

When an IBM Informix product accesses an environment variable, normally the following rules of precedence apply:

1. Of highest precedence is the value that is defined in the environment (shell) by explicitly setting the value at the shell prompt.
2. The second highest precedence goes to the value that is defined in the private environment-configuration file in the home directory of the user (~/.informix).
3. The next highest precedence goes to the value that is defined in the common environment-configuration file ($INFORMIXDIR/etc/informix.rc).
4. The lowest precedence goes to the default value, if one exists.

For precedence information about GLS environment variables, see the IBM Informix GLS User’s Guide.

**Important:** If you set one or more environment variables before you start the database server, and you do not explicitly set the same environment variables for your client products, the clients will adopt the original settings.
Using Environment Variables on Windows

The following sections discuss setting, viewing, unsetting, and modifying environment variables for Windows® applications.

Where to Set Environment Variables on Windows

You can set environment variables in several places on Windows, depending on which IBM Informix application you use.

Environment variables can be set in several ways, as described in “Environment Settings” on page 3-8.

The SET ENVIRONMENT statement of SQL can set certain routine-specific environment options. For more information, refer to the description of SET ENVIRONMENT in the IBM Informix Guide to SQL: Syntax.

To use client applications such as ESQL/C or the Schema Tools on Windows environment, use the Setnet32 utility to set environment variables. For information about the Setnet32 utility, see the IBM Informix Client Products Installation Guide for your operating system.

In IBM Informix ESQL/C, you can set supported environment variables within an application with the ifx_putenv() function and retrieve values with the ifx_getenv() function, if your Windows system supports them. For more information on ifx_putenv() and ifx_getenv(), see the IBM Informix ESQL/C Programmer's Manual.

Environment Settings

You can set environment variables for command-prompt utilities in the following ways:

• With the System applet in the Control Panel

• In a command-line session

Using the System Applet to Change Environment Variables

The System applet provides a graphical interface to create, modify, and delete system-wide and user-specific variables. Environment variables that are set with the System applet are visible to all command-prompt sessions.

To change environment variables with the System applet in the control panel:

1. Double-click the System applet icon from the Control Panel window.

2. Click the Environment tab near the top of the window. Two list boxes display System Environment Variables and User Environment Variables. System Environment Variables apply to an entire system, and User Environment Variables apply only to the sessions of the individual user.

3. To change the value of an existing variable, select that variable.

4. Highlight the existing value and type the new value.

5. Next, enter the value for the new variable at the bottom of the window and click the Set button.
6. To delete a variable, select the variable and click the **Delete** button.

**Important:** In order to use the System applet to change System environment variables, you must belong to the Administrators group. For information on assigning users to groups, see your operating-system documentation.

**Using the Command Prompt to Change Environment Variables**

The following diagram shows the syntax for setting an environment variable at a command prompt in Windows.

```
set variable=value
```

If no value is specified, the environment variable is unset, as if it did not exist.

For more information on how to read syntax diagrams, see “Syntax Diagrams” on page xv of the introduction.

To view your current settings after one or more IBM Informix products are installed, enter the following command at the command prompt.

```
set
```

Sometimes you must add information to an environment variable that is already set. For example, the **PATH** environment variable is always set in Windows environments. When you use an IBM Informix product, you must add the name of the directory where the executable files for the IBM Informix products are stored to the **PATH**.

In the following example, **INFORMIXDIR** is **d:\informix** (that is, during installation, IBM Informix products were installed in the **d:\informix** directory). The executable files are in the **bin** subdirectory, **d:\informix\bin**. To add this directory at the beginning of the **PATH** environment-variable value, use the following command:

```
set PATH=d:\informix\bin;%PATH%
```

Rather than entering an explicit pathname, you can use the value of the **INFORMIXDIR** environment variable (represented as **%INFORMIXDIR%**), as the following example shows:

```
set INFORMIXDIR=d:\informix
set PATH=%INFORMIXDIR%\bin;%PATH%
```

You might prefer to use this version to ensure that your **PATH** entry does not contradict the search path that was set in **INFORMIXDIR** and to avoid the need to reset **PATH** whenever you change **INFORMIXDIR**.

For more information about setting and modifying environment variables, refer to your operating-system publications.

**Using dbservername.cmd to Initialize a Command-Prompt Environment**

Each time that you open a Windows command prompt, it acts as an independent environment. Therefore, environment variables that you set within it are valid only for that particular command-prompt instance.
For example, if you open one command window and set the variable, `INFORMIXDIR`, and then open another command window and type `set` to check your environment, you will find that `INFORMIXDIR` is not set in the new command-prompt session.

The database server installation program creates a batch file that you can use to configure command-prompt utilities, ensuring that your command-prompt environment is initialized correctly each time that you run a command-prompt session. The batch file, `dbservername.cmd`, is located in `%INFORMIXDIR%`, and is a plain text file that you can modify with any text editor. If you have more than one database server installed in `%INFORMIXDIR%`, there will be more than one batch file with the `.cmd` extension, each bearing the name of the database server with which it is associated.

To run `dbservername.cmd` from a command prompt, type `dbservername` or configure a command prompt so that it runs `dbservername.cmd` automatically at start up.

**Rules of Precedence**

When an IBM Informix product accesses an environment variable, normally the following rules of precedence apply:

1. The setting in Setnet32 with the **Use my settings** box selected.
2. The setting in Setnet32 with the **Use my settings** box cleared.
3. The setting on the command line before running the application.
4. The setting in Windows as a user variable.
5. The setting in Windows as a system variable.
6. The lowest precedence goes to the default value.

An application examines the first five values as it starts. Unless otherwise stated, changing an environment variable after the application is running does not have any effect.

**List of Environment Variables**

Table 3-1 contains an alphabetical list of the environment variables that you can set for an Informix database server and SQL API products. Most of these environment variables are described in this chapter on the pages listed in the “Page” column. The • symbol indicates that XPS or Dynamic Server (or both, if both columns are so marked) support the environment variable.

The notation ERG in the Page column indicates an environment variable that must be set with the CDR_ENV configuration parameter and that is described in the appendix on configuration parameters and environment variables of the *IBM Informix Dynamic Server Enterprise Replication Guide*.

The notation GLS in the Page column indicates a GLS environment variable that is valid in nondefault locales and that is described in the GLS environment variables chapter of *IBM Informix GLS User’s Guide*.

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>XPS</th>
<th>IDS</th>
<th>Restrictions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC_CONFIG</td>
<td>X</td>
<td>X</td>
<td>ON-Bar</td>
<td>3-14</td>
</tr>
<tr>
<td>AFDEBUG</td>
<td></td>
<td></td>
<td>JVM</td>
<td>3-14</td>
</tr>
</tbody>
</table>
### Table 3-1. Alphabetical List of Environment Variables (continued)

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>XPS</th>
<th>IDS</th>
<th>Restrictions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSIOWNER</td>
<td>X</td>
<td>None</td>
<td>None</td>
<td>3-14</td>
</tr>
<tr>
<td>BIG_FET_BUF_SIZE</td>
<td>X</td>
<td>SQL APIs and DB-Access only</td>
<td>3-15</td>
<td></td>
</tr>
<tr>
<td>CC8BITLEVEL</td>
<td></td>
<td>ESQL/C only</td>
<td>GLS</td>
<td></td>
</tr>
<tr>
<td>CDRSITES_731</td>
<td>X</td>
<td>ER only</td>
<td>ERG</td>
<td></td>
</tr>
<tr>
<td>CDRSITES_92X</td>
<td>X</td>
<td>ER only</td>
<td>ERG</td>
<td></td>
</tr>
<tr>
<td>CDR_LOGDELTA</td>
<td>X</td>
<td>ER only</td>
<td>ERG</td>
<td></td>
</tr>
<tr>
<td>CDR_PERFLOG</td>
<td>X</td>
<td>ER only</td>
<td>ERG</td>
<td></td>
</tr>
<tr>
<td>CDR_ROUTER</td>
<td>X</td>
<td>ER only</td>
<td>ERG</td>
<td></td>
</tr>
<tr>
<td>CDR_RMSCALEFACT</td>
<td>X</td>
<td>ER only</td>
<td>ERG</td>
<td></td>
</tr>
<tr>
<td>CLIENT_LOCALE</td>
<td>X</td>
<td>None</td>
<td>GLS</td>
<td></td>
</tr>
<tr>
<td>CPFIRST</td>
<td>X</td>
<td>ESQL/C only</td>
<td>3-15</td>
<td></td>
</tr>
<tr>
<td>DBACCNOIGN</td>
<td>X</td>
<td>DB-Access only</td>
<td>3-16</td>
<td></td>
</tr>
<tr>
<td>DBANSIWARN</td>
<td>X</td>
<td>None</td>
<td>3-17</td>
<td></td>
</tr>
<tr>
<td>DBBLOBBUFF</td>
<td>X</td>
<td>UNLOAD only</td>
<td>3-17</td>
<td></td>
</tr>
<tr>
<td>DBCENTURY</td>
<td></td>
<td>SQL APIs only</td>
<td>3-18</td>
<td></td>
</tr>
<tr>
<td>DBDATE</td>
<td>X</td>
<td>None</td>
<td>3-20</td>
<td></td>
</tr>
<tr>
<td>DBDELMITTER</td>
<td>X</td>
<td>None</td>
<td>3-22</td>
<td></td>
</tr>
<tr>
<td>DBEDIT</td>
<td>X</td>
<td>None</td>
<td>3-22</td>
<td></td>
</tr>
<tr>
<td>DBFLTMASK</td>
<td>X</td>
<td>DB-Access only</td>
<td>3-23</td>
<td></td>
</tr>
<tr>
<td>DBLANG</td>
<td>X</td>
<td>None</td>
<td>3-23</td>
<td></td>
</tr>
<tr>
<td>DBMONEY</td>
<td>X</td>
<td>None</td>
<td>3-24</td>
<td></td>
</tr>
<tr>
<td>DBNLS</td>
<td>X</td>
<td></td>
<td>3-25</td>
<td></td>
</tr>
<tr>
<td>DBONUPLOAD</td>
<td>X</td>
<td>HPL only</td>
<td>3-26</td>
<td></td>
</tr>
<tr>
<td>DBPATH</td>
<td>X</td>
<td>None</td>
<td>3-26</td>
<td></td>
</tr>
<tr>
<td>DBPRINT</td>
<td>X</td>
<td>UNIX only</td>
<td>3-28</td>
<td></td>
</tr>
<tr>
<td>DBREMOTECMD</td>
<td>X</td>
<td>UNIX only</td>
<td>3-28</td>
<td></td>
</tr>
<tr>
<td>DBSPACETEMP</td>
<td>X</td>
<td>None</td>
<td>3-29</td>
<td></td>
</tr>
<tr>
<td>DBTEMP</td>
<td>X</td>
<td>DB-Access, Gateways</td>
<td>3-30</td>
<td></td>
</tr>
<tr>
<td>DBTIME</td>
<td></td>
<td>SQL APIs only</td>
<td>3-31</td>
<td></td>
</tr>
<tr>
<td>DBUPSPACE</td>
<td>X</td>
<td>None</td>
<td>3-33</td>
<td></td>
</tr>
<tr>
<td>DB_LOCALE</td>
<td>X</td>
<td>None</td>
<td>GLS</td>
<td></td>
</tr>
<tr>
<td>DEFAULT_ATTACH</td>
<td>X</td>
<td>Deprecated</td>
<td>3-34</td>
<td></td>
</tr>
<tr>
<td>DELIMIDENT</td>
<td>X</td>
<td>None</td>
<td>3-34</td>
<td></td>
</tr>
<tr>
<td>ENVIGNORE</td>
<td>X</td>
<td>UNIX only</td>
<td>3-35</td>
<td></td>
</tr>
<tr>
<td>ESQLMF</td>
<td>X</td>
<td>ESQL/C only</td>
<td>GLS</td>
<td></td>
</tr>
<tr>
<td>FET_BUF_SIZE</td>
<td>X</td>
<td>SQL APIs, DB-Access only</td>
<td>3-36</td>
<td></td>
</tr>
<tr>
<td>GLOBAL_DETACH_INFORM</td>
<td>X</td>
<td>None</td>
<td>3-36</td>
<td></td>
</tr>
<tr>
<td>GLS8BITFSYS</td>
<td>X</td>
<td>None</td>
<td>GLS</td>
<td></td>
</tr>
<tr>
<td>GL_DATE</td>
<td>X</td>
<td>None</td>
<td>GLS</td>
<td></td>
</tr>
</tbody>
</table>

Chapter 3. Environment Variables  3-11
<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>XPS</th>
<th>IDS</th>
<th>Restrictions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GL_DATETIME</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>GLS</td>
</tr>
<tr>
<td>IBM_XPS_PARAMS</td>
<td>X</td>
<td>None</td>
<td>None</td>
<td>3-37</td>
</tr>
<tr>
<td>IFMX_CART_ALRM</td>
<td>X</td>
<td>None</td>
<td>None</td>
<td>3-38</td>
</tr>
<tr>
<td>IFMX_HISTORY_SIZE</td>
<td>X</td>
<td>DB-Access</td>
<td>None</td>
<td>3-38</td>
</tr>
<tr>
<td>IFMX_OPT_FACT_TABS</td>
<td>X</td>
<td>None</td>
<td>None</td>
<td>3-38</td>
</tr>
<tr>
<td>IFMX_OPT_NON_DIM_TABS</td>
<td>X</td>
<td>None</td>
<td>None</td>
<td>3-39</td>
</tr>
<tr>
<td>IFX_DEF_TABLE_LOCKMODE</td>
<td>X</td>
<td>None</td>
<td>None</td>
<td>3-39</td>
</tr>
<tr>
<td>IFX_DIRECTIONS</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-40</td>
</tr>
<tr>
<td>IFX_EXTDIRECTIVES</td>
<td>X</td>
<td>X</td>
<td>Set on the client only</td>
<td>3-41</td>
</tr>
<tr>
<td>IFX_LONGID</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-41</td>
</tr>
<tr>
<td>IFX_NETBUF_PVTPOOL_SIZE</td>
<td>X</td>
<td>X</td>
<td>UNIX only</td>
<td>3-42</td>
</tr>
<tr>
<td>IFX_NETBUF_SIZE</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-42</td>
</tr>
<tr>
<td>IFX_NO_LRU_TUNING</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-43</td>
</tr>
<tr>
<td>IFX_NO_TIMELIMIT_WARNING</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-43</td>
</tr>
<tr>
<td>IFX_NODBPROC</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-43</td>
</tr>
<tr>
<td>IFX_ONTAPE_FILE_PREFIX</td>
<td>X</td>
<td>None</td>
<td>None</td>
<td>3-44</td>
</tr>
<tr>
<td>IFX_PAD_VARCHAR</td>
<td>X</td>
<td>None</td>
<td>None</td>
<td>3-44</td>
</tr>
<tr>
<td>IFX_UPDDESC</td>
<td>X</td>
<td>None</td>
<td>None</td>
<td>3-44</td>
</tr>
<tr>
<td>IFX_XASTDCOMPLIANCE_XAEND</td>
<td>X</td>
<td>None</td>
<td>None</td>
<td>3-45</td>
</tr>
<tr>
<td>IFX_XFER_SHMBASE</td>
<td>X</td>
<td>None</td>
<td>None</td>
<td>3-45</td>
</tr>
<tr>
<td>IMCADMIN</td>
<td>X</td>
<td></td>
<td></td>
<td>3-45</td>
</tr>
<tr>
<td>IMCCONFIG</td>
<td>X</td>
<td></td>
<td></td>
<td>3-46</td>
</tr>
<tr>
<td>IMCSERVER</td>
<td>X</td>
<td></td>
<td></td>
<td>3-46</td>
</tr>
<tr>
<td>INFORMIXC</td>
<td></td>
<td></td>
<td>ESQL/C, UNIX only</td>
<td>3-47</td>
</tr>
<tr>
<td>INFORMIXCONCSMCFG</td>
<td>X</td>
<td>None</td>
<td>None</td>
<td>3-47</td>
</tr>
<tr>
<td>INFORMIXCONRETRY</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-47</td>
</tr>
<tr>
<td>INFORMIXCONTIME</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-48</td>
</tr>
<tr>
<td>INFORMIXCPPMAP</td>
<td>X</td>
<td>None</td>
<td>None</td>
<td>3-49</td>
</tr>
<tr>
<td>INFORMIXDIR</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-49</td>
</tr>
<tr>
<td>INFORMIXKEYTAB</td>
<td>X</td>
<td>X</td>
<td>UNIX only</td>
<td>3-49</td>
</tr>
<tr>
<td>INFORMIXOPCACHE</td>
<td>X</td>
<td>None</td>
<td>Optical Subsystem only</td>
<td>3-50</td>
</tr>
<tr>
<td>INFORMIXSERVER</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-50</td>
</tr>
<tr>
<td>INFORMIXSHMBASE</td>
<td>X</td>
<td>X</td>
<td>UNIX only</td>
<td>3-51</td>
</tr>
<tr>
<td>INFORMIXSQLHOSTS</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-51</td>
</tr>
<tr>
<td>INFORMIXSTACKSIZE</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-52</td>
</tr>
<tr>
<td>INFORMIXTERM</td>
<td>X</td>
<td>X</td>
<td>DB-Access, UNIX only</td>
<td>3-52</td>
</tr>
<tr>
<td>INF_ROLE_SEP</td>
<td>X</td>
<td>None</td>
<td>None</td>
<td>3-53</td>
</tr>
<tr>
<td>INTERACTIVE_DESKTOP_OFF</td>
<td>X</td>
<td></td>
<td>Windows only</td>
<td>3-54</td>
</tr>
<tr>
<td>ISM_COMPRESSION</td>
<td>X</td>
<td>X</td>
<td>ISM, ON-Bar only</td>
<td>3-54</td>
</tr>
<tr>
<td>ISM_DEBUG_FILE</td>
<td>X</td>
<td>X</td>
<td>ISM only</td>
<td>3-54</td>
</tr>
</tbody>
</table>
### Table 3-1. Alphabetical List of Environment Variables (continued)

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>XPS</th>
<th>IDS</th>
<th>Restrictions</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISM_DEBUG_LEVEL</td>
<td>X</td>
<td>X</td>
<td>ISM, ON–Bar only</td>
<td>3-55</td>
</tr>
<tr>
<td>ISM_ENCRYPTION</td>
<td>X</td>
<td>X</td>
<td>ISM, ON–Bar only</td>
<td>3-55</td>
</tr>
<tr>
<td>ISM_MAXLOGSIZE</td>
<td>X</td>
<td>X</td>
<td>ISM only</td>
<td>3-55</td>
</tr>
<tr>
<td>ISM_MAXLOGVERS</td>
<td>X</td>
<td>X</td>
<td>ISM only</td>
<td>3-55</td>
</tr>
<tr>
<td>JAR_TEMP_PATH</td>
<td>X</td>
<td></td>
<td>JVM</td>
<td>3-56</td>
</tr>
<tr>
<td>JAVA_COMPILER</td>
<td></td>
<td></td>
<td>JVM</td>
<td>3-56</td>
</tr>
<tr>
<td>JVM_MAX_HEAP_SIZE</td>
<td></td>
<td></td>
<td>JVM</td>
<td>3-56</td>
</tr>
<tr>
<td>LD_LIBRARY_PATH</td>
<td></td>
<td></td>
<td>SQL APIs, UNIX only</td>
<td>3-57</td>
</tr>
<tr>
<td>LIBERAL_MATCH</td>
<td>X</td>
<td></td>
<td>None</td>
<td>3-57</td>
</tr>
<tr>
<td>LIBPATH</td>
<td></td>
<td></td>
<td>SQL APIs, UNIX only</td>
<td>3-57</td>
</tr>
<tr>
<td>NODEFDAC</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-58</td>
</tr>
<tr>
<td>ONCONFIG</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-58</td>
</tr>
<tr>
<td>OPTCOMPIND</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-59</td>
</tr>
<tr>
<td>OPTMSG</td>
<td></td>
<td></td>
<td>ESQ/C only</td>
<td>3-59</td>
</tr>
<tr>
<td>OPTOFC</td>
<td></td>
<td></td>
<td>ESQ/C only</td>
<td>3-60</td>
</tr>
<tr>
<td>OPT_GOAL</td>
<td>X</td>
<td></td>
<td>UNIX only</td>
<td>3-60</td>
</tr>
<tr>
<td>PATH</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-61</td>
</tr>
<tr>
<td>PDQPRIORITY</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-61</td>
</tr>
<tr>
<td>PLC_CONFIG</td>
<td>X</td>
<td></td>
<td>HPL only</td>
<td>3-62</td>
</tr>
<tr>
<td>PLOAD_LO_PATH</td>
<td>X</td>
<td></td>
<td>HPL only</td>
<td>3-63</td>
</tr>
<tr>
<td>PLOAD_SHMBASE</td>
<td>X</td>
<td></td>
<td>HPL only</td>
<td>3-63</td>
</tr>
<tr>
<td>PSORT_DBTEMP</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-63</td>
</tr>
<tr>
<td>PSORT_NPROCS</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>3-64</td>
</tr>
<tr>
<td>RTREE_COST_ADJUST_VALUE</td>
<td>X</td>
<td></td>
<td>None</td>
<td>3-65</td>
</tr>
<tr>
<td>SERVER_LOCALE</td>
<td>X</td>
<td>X</td>
<td>None</td>
<td>GLS</td>
</tr>
<tr>
<td>SHLIB_PATH</td>
<td>X</td>
<td>X</td>
<td>UNIX only</td>
<td>3-65</td>
</tr>
<tr>
<td>STMT_CACHE</td>
<td>X</td>
<td></td>
<td>None</td>
<td>3-66</td>
</tr>
<tr>
<td>TERM</td>
<td>X</td>
<td>X</td>
<td>UNIX only</td>
<td>3-66</td>
</tr>
<tr>
<td>TERMCAP</td>
<td>X</td>
<td>X</td>
<td>UNIX only</td>
<td>3-66</td>
</tr>
<tr>
<td>TERMINFO</td>
<td>X</td>
<td>X</td>
<td>UNIX only</td>
<td>3-67</td>
</tr>
<tr>
<td>THREADLIB</td>
<td></td>
<td></td>
<td>ESQ/L, UNIX only</td>
<td>3-68</td>
</tr>
<tr>
<td>TOBIGINT</td>
<td>X</td>
<td></td>
<td>dbschema only</td>
<td>3-68</td>
</tr>
<tr>
<td>USE_DTENV</td>
<td>X</td>
<td></td>
<td>None</td>
<td>IBM Informix ESQ/L/C Programmer’s Manual</td>
</tr>
<tr>
<td>USETABLENAME</td>
<td>X</td>
<td></td>
<td>None</td>
<td>3-68</td>
</tr>
<tr>
<td>XFER_CONFIG</td>
<td>X</td>
<td></td>
<td>None</td>
<td>3-68</td>
</tr>
</tbody>
</table>

Tip: You might encounter references to environment variables that are not listed in Table 3-1 on page 3-10. Most likely, these environment variables are not
supported in this release or are used to maintain backward compatibility with certain earlier product versions. For information, refer to an earlier version of your IBM Informix documentation.

**Environment Variables**

Sections that follow discuss (in alphabetical order) environment variables that IBM Informix database server products and their utilities use.

**Important:** The descriptions of the following environment variables include the syntax for setting the environment variable on UNIX. For a general description of how to set these environment variables on Windows, see "Environment Settings" on page 3-8.

**AC_CONFIG**

You can set the **AC_CONFIG** environment variable to specify the path for the **ac_config.std** configuration file for the **archecker** utility, which checks the validity and completeness of an ON-Bar storage-space backup. The **ac_config.std** file contains default **archecker** configuration parameters.

```bash
setenv--AC_CONFIG--pathname
```

*pathname* is the location of the **ac_config.std** configuration file in `$INFORMIXDIR/etc` or `%INFORMIXDIR%/etc`.

For information on **archecker**, see your IBM Informix Backup and Restore Guide.

**AFDEBUG**

You can create files to hold verbose messages from the Java virtual machine (JVM) about releasing memory that had been allocated to objects by setting the **AFDEBUG** environment variable.

```bash
setenv--AFDEBUG
```

No value is required. You can also set the configuration parameter **AFCRASH** to 0x00000010 to achieve the same result.

**ANSIOWNER (IDS)**

In an ANSI-compliant database, you can prevent the default behavior of upshifting lowercase letters in owner names that are not delimited by quotation marks by setting the **ANSIOWNER** environment variable to 1.

```bash
setenv--ANSIOWNER--1
```

To prevent upshifting of lowercase letters in owner names in an ANSI-compliant database, you must set **ANSIOWNER** before you initialize Dynamic Server.

The following table shows how an ANSI-compliant database of Dynamic Server stores or reads the specified name of a database object called **oblong** if you were the owner of **oblong** and your **userid** (in all lowercase letters) were **owen**:
Because they do not match the lettercase of your userid, any SQL statements that specified the formats that are stored as OWEN.oblong would fail with errors.

**BIG_FET_BUF_SIZE (XPS)**

The BIG_FET_BUF_SIZE environment variable functions the same as the FET_BUF_SIZE environment variable, but supports a larger cursor buffer.

```
setenv BIG_FET_BUF_SIZE size
```

size is a positive integer that is larger than the default buffer size.

The size can be no greater than 4 gigabytes and specifies the size (in bytes) of the fetch buffer that holds data retrieved by a query. For example, to set a buffer size to 5,000 bytes on a UNIX system that uses the C shell, set the BIG_FET_BUF_SIZE environment variable with the following command:

```
setenv BIG_FET_BUF_SIZE 5000
```

When BIG_FET_BUF_SIZE is set to a valid value, the new value overrides the default value (or any previously set value of BIG_FET_BUF_SIZE). The default setting for the fetch buffer is dependent on row size. The processing of BYTE and TEXT values is not affected by BIG_FET_BUF_SIZE.

No error is raised if BIG_FET_BUF_SIZE is set to a value less than the default size or out of the range of SMALLINT values. In these cases, however, the invalid fetch buffer size is ignored and the default size is in effect.

If you set BIG_FET_BUF_SIZE to a valid value, that value is in effect for the local database server as well as for any remote database server from which you retrieve rows through a distributed query in which the local server is the coordinator and the remote server is subordinate. The greater the size of the buffer, the more rows can be returned and the less frequently the client application must wait for returned rows. A large buffer can improve performance by reducing the overhead of filling the client-side buffer.

**CPFIRST**

Set the CPFIRST environment variable to specify the default compilation order for all ESQL/C source files in your programming environment.

```
setenv CPFIRST TRUE
```

When you compile an ESQL/C program with CPFIRST not, set the ESQL/C preprocessor runs first, by default, on the program source file and then passes the resulting file to the C language preprocessor and compiler. You can, however, compile an ESQL/C program source file in the following order:
1. Run the C preprocessor
2. Run the ESQL/C preprocessor
3. Run the C compiler and linker

To use a nondefault compilation order for a specific program, you can either give the program source file a .ecp extension, run the -cp option with the esql command on a program source file with a .ec extension, or set CPFIRST.

Set CPFIRST to TRUE (uppercase only) to run the C preprocessor before the ESQL/C preprocessor on all ESQL/C source files in your environment, irrespective of whether the -cp option is passed to the esql command or the source files have the .ec or the .ecp extension.

To restore the default order on a system where the CPFIRST environment variable has been set to TRUE, you can set CPFIRST to FALSE. On UNIX systems that support the C shell, the following command has the same effect:

unsetenv CPFIRST

DBACCNOIGN

The DBACCNOIGN environment variable affects the behavior of the DB–Access utility if an error occurs under one of the following circumstances:

- You run DB–Access in nonmenu mode.
- In Dynamic Server only, you execute the LOAD command with DB–Access in menu mode.

Set the DBACCNOIGN environment variable to 1 to roll back an incomplete transaction if an error occurs while you run the DB–Access utility under either of the preceding conditions.

>>setenv DBACCNOIGN 1

For example, assume DB–Access runs the following SQL commands:

DATABASE mystore
BEGIN WORK

INSERT INTO receipts VALUES (cust1, 10)
INSERT INTO receipt VALUES (cust1, 20)
INSERT INTO receipts VALUES (cust1, 30)

UPDATE customer
SET balance =
(SELECT balance-60)
FROM customer WHERE custid = 'cust1'
WHERE custid = 'cust1'
COMMIT WORK

Here, one statement has a misspelled table name: the receipt table does not exist. If DBACCNOIGN is not set in your environment, DB–Access inserts two records into the receipts table and updates the customer table. Now, the decrease in the customer balance exceeds the sum of the inserted receipts.

But if DBACCNOIGN is set to 1, messages appear that indicate that DB–Access rolled back all the INSERT and UPDATE statements. The messages also identify the cause of the error so that you can resolve the problem.
LOAD Statement Example
You can set DBACCNOIGN to protect data integrity during a LOAD statement, even if DB–Access runs the LOAD statement in menu mode.

Assume you execute the LOAD statement from the DB–Access SQL menu. Forty-nine rows of data load correctly, but the 50th row contains an invalid value that causes an error. If you set DBACCNOIGN to 1, the database server does not insert the forty-nine previous rows into the database. If DBACCNOIGN is not set, the database server inserts the first forty-nine rows.

DBANSIWARN
Setting the DBANSIWARN environment variable indicates that you want to check for Informix extensions to ANSI-standard SQL syntax. Unlike most environment variables, you do not need to set DBANSIWARN to a value. You can set it to any value or to no value.

```bash
>>-setenv=DBANSIWARN
```

Running DB–Access with DBANSIWARN set is functionally equivalent to including the -ansi flag when you invoke DB–Access (or any IBM Informix product that recognizes the -ansi flag) from the command line. If you set DBANSIWARN before you run DB–Access, any syntax-extension warnings are displayed on the screen within the SQL menu.

At runtime, the DBANSIWARN environment variable causes the sixth character of the sqlwarn array in the SQL Communication Area (SQLCA) to be set to 1 when a statement is executed that is recognized as including any Informix extension to the ANSI/ISO standard for SQL syntax.

For details on SQLCA, see the IBM Informix ESQL/C Programmer’s Manual.

After you set DBANSIWARN, Informix extension checking is automatic until you log out or unset DBANSIWARN. To turn off Informix extension checking, you can disable DBANSIWARN with this command:

```bash
unsetenv DBANSIWARN
```

DBBLOBBUF
The DBBLOBBUF environment variable controls whether TEXT or BYTE values are stored temporarily in memory or in a file while being processed by the UNLOAD statement. DBBLOBBUF affects only the UNLOAD statement.

```bash
>>-setenv=DBBLOBBUF-size
```

`size` represents the maximum size of TEXT or BYTE data in kilobytes.

If the TEXT or BYTE data size is smaller than the default of 10 kilobytes (or the setting of DBBLOBBUF), the TEXT or BYTE value is temporarily stored in memory. If the data size is larger than the default or the DBBLOBBUF setting, the data value is written to a temporary file. For instance, to set a buffer size of 15 kilobytes, set DBBLOBBUF as in the following example:

```bash
setenv DBBLOBBUF 15
```
Here any TEXT or BYTE value smaller than 15 kilobytes is stored temporarily in memory. Values larger than 15 kilobytes are stored temporarily in a file.

**DBCENTURY**

To avoid problems in expanding abbreviated years, applications should require entry of 4-digit years, and should always display years as four digits. The **DBCENTURY** environment variable specifies how to expand literal DATE and DATETIME values that are entered with abbreviated year values.

When **DBCENTURY** is not set (or is set to R), the first two digits of the current year are used to expand 2-digit year values. For example, if today’s date is 09/30/2003, then the abbreviated date 12/31/99 expands to 12/31/2099, and the abbreviated date 12/31/00 expands to 12/31/2000.

The R, P, F, and C settings choose algorithms for expanding two-digit years.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>R = Current</td>
<td>Use the first two digits of the current year to expand the year value.</td>
</tr>
<tr>
<td>P = Past</td>
<td>Expanded dates are created by prefixing the abbreviated year value with 19 and 20. Both dates are compared to the current date, and the most recent date that is earlier than the current date is used.</td>
</tr>
<tr>
<td>F = Future</td>
<td>Expanded dates are created by prefixing the abbreviated year value with 20 and 21. Both dates are compared to the current date, and the earliest date that is later than the current date is used.</td>
</tr>
<tr>
<td>C = Closest</td>
<td>Expanded dates are created by prefixing the abbreviated year value with 19, 20, and 21. These three dates are compared to the current date, and the date that is closest to the current date is used.</td>
</tr>
</tbody>
</table>

Settings are case sensitive, and no error is issued for invalid settings. If you enter f (for example), then the default (R) setting takes effect. The P and F settings cannot return the current date, which is not in the past or future.

Years entered as a single digit are prefixed with 0 and then expanded. Three-digit years are not expanded. Pad years earlier than 100 with leading zeros.

**Examples of Expanding Year Values**
The following examples illustrate how various settings of **DBCENTURY** cause abbreviated years to be expanded in DATE and DATETIME values.

**DBCENTURY = P:**

Example data type: DATE  
Current date: 4/6/2003  
User enters: 1/1/1  
Prefix with "19" expansion: 1/1/1901  
Prefix with "20" expansion: 1/1/2001  
Analysis: Both are prior to current date, but 1/1/2001 is closer to current date.
Important: The effect of DBCENTURY depends on the current date from the system clock-calendar. Thus, 1/1/1, the abbreviated date in this example, would instead be expanded to 1/1/1901 if the current date were 1/1/2001 and DBCENTURY = P.

DBCENTURY = F:
Example data type: DATETIME year to month
Current date: 5/7/2005
User enters: 1-1
Prefix with "20" expansion: 2001-1
Prefix with "21" expansion: 2101-1
Analysis: Only date 2101-1 is after the current date, so it is chosen.

DBCENTURY = C:
Example data type: DATE
Current date: 4/6/2000
User enters: 1/1/1
Prefix with "19" expansion: 1/1/1901
Prefix with "20" expansion: 1/1/2001
Prefix with "21" expansion: 1/1/2101
Analysis: Here 1/1/2001 is closest to the current date, so it is chosen.

DBCENTURY = R or DBCENTURY Not Set:
Example data type: DATETIME year to month
Current date: 4/6/2000
User enters: 1-1
Prefix with "20" expansion: 2001-1

Example data type: DATE
Current date: 4/6/2003
User enters: 0/1/1
Prefix with "20" expansion: 2000/1
Analysis: In both examples, the Prefix with "20" algorithm is used.

Setting DBCENTURY does not affect IBM Informix products when the locale specifies a non-Gregorian, calendar such as Hebrew or Islamic calendars. The leading digits of the current year are used for alternate calendar systems when the year is abbreviated.

Abbreviated Years and Expressions in Database Objects
When an expression in a database object (including a check constraint, fragmentation expression, SPL routine, trigger, or UDR) contains a literal date or DATETIME value in which the year has one or two digits, the database server evaluates the expression using the setting that DBCENTURY (and other relevant environment variables) had when the database object was created (or was last modified). If DBCENTURY has been reset to a new value, the new value is ignored when the abbreviated year is expanded.

For example, suppose a user creates a table and defines the following check constraint on a column named birthdate:

```
birthdate < '09/25/50'
```

The expression is interpreted according to the value of DBCENTURY when the constraint was defined. If the table contains the birthdate column is created on 09/23/2000 and DBCENTURY = C, the check constraint expression is consistently interpreted as birthdate < '09/25/1950' when inserts or updates are performed on the birthdate column. Even if different values of DBCENTURY are set when users perform inserts or updates on the birthdate column, the constraint
expression is interpreted according to the setting at the time when the check
constraint was defined (or was last modified).

Database objects created on some earlier versions of Dynamic Server do not
support the priority of creation-time settings.

For legacy objects to acquire this feature:
1. Drop the objects.
2. Re-create them (or for fragmentation expressions, detach them and then
   reattach them).

After the objects are redefined, date literals within expressions of the objects will
be interpreted according to the environment at the time when the object was
created or was last modified. Otherwise, their behavior will depend on the runtime
environment and might become inconsistent if this changes.

Administration of a database that includes a mix of legacy objects and new objects
might become difficult because of differences between the new and the old
behavior for evaluating date expressions. To avoid this, it is recommended that you
redfine any legacy objects.

The value of DBCENTURY and the current date are not the only factors that
determine how the database server interprets date and DATETIME values. The
DBDATE, DBTIME, GL_DATE, and GL_DATETIME environment variables can
also influence how dates are interpreted. For information about GL_DATE and
GL_DATETIME, see the IBM Informix GLS User’s Guide.

Important: The behavior of DBCENTURY for Dynamic Server and Extended
Parallel Server is not backwards compatible.

DBDATE

The DBDATE environment variable specifies the end-user formats of DATE values.
On UNIX systems that use the C shell, set DBDATE with this syntax.

```
setenv DBDATE DM D M Y4 Y2 / 0
```

The following formatting symbols are valid in the DBDATE setting:
- . / are characters that can appear as separators in a date format.
- 0 indicates that no separator is displayed between time units.
- D, M are characters that represent the day and the month.
- Y2, Y4 are characters that represent the year and the precision of the year.

Some East Asian locales support additional syntax for era-based dates. For details
of era-based formats, see IBM Informix GLS User’s Guide.

DBDATE can specify the following attributes of the display format:
- The order of time units (the month, day, and year) in a date
- Whether the year appears with two digits (Y2) or four digits (Y4)
• The separator between the month, day, and year time units

For the U.S. English locale, the default for DBDATE is MDY4/, where M represents the month, D represents the day, Y4 represents a four-digit year, and slash (/) is the time-units separator (for example, 01/08/2002). Other valid characters for the separator are a hyphen (-), a period (.), or a zero (0). To indicate no separator, use the zero. The slash (/) is used by default if you attempt to specify a character other than a hyphen, period, or zero as a separator, or if you do not include any separator in the DBDATE specification.

If DBDATE is not set on the client, any DBDATE setting on the database server overrides the MDY4/ default on the client. If DBDATE is set on the client, that value (rather than the setting on the database server) is used by the client.

The following table shows some examples of valid DBDATE settings and their corresponding displays for the date 8 January, 2005:

<table>
<thead>
<tr>
<th>DBDATE Setting</th>
<th>Representation of January 8, 2005:</th>
<th>DBDATE Setting</th>
<th>Representation of January 8, 2005:</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDY4/</td>
<td>01/08/2005</td>
<td>Y2DM.</td>
<td>05.08.01</td>
</tr>
<tr>
<td>DMY2-</td>
<td>08-01-05</td>
<td>MDY20</td>
<td>010805</td>
</tr>
<tr>
<td>MDY4</td>
<td>01/08/2005</td>
<td>Y4MD*</td>
<td>2005/01/08</td>
</tr>
</tbody>
</table>

Formats Y4MD* (because asterisk is not a valid separator) and MDY4 (with no separator defined) both display the default symbol (slash) as the separator.

Important: If you use the Y2 format, the setting of the DBCENTURY environment variable can also affect how literal DATE values are evaluated in data entry.

Also, certain routines that IBM Informix ESQL/C calls can use the DBTIME variable, rather than DBDATE, to set DATETIME formats to international specifications. For more information, see the discussion of the DBTIME environment variable in [“DBTIME” on page 3-31] and in the IBM Informix ESQL/C Programmer’s Manual.

The setting of the DBDATE variable takes precedence over that of the GL_DATE environment variable, as well as over any default DATE format that CLIENT_LOCALE specifies. For information about GL_DATE and CLIENT_LOCALE, see the IBM Informix GLS User’s Guide.

End-user formats affect the following contexts:
• When you display DATE values, IBM Informix products use the DBDATE environment variable to format the output.
• During data entry of DATE values, IBM Informix products use the DBDATE environment variable to interpret the input.

For example, if you specify a literal DATE value in an INSERT statement, the database server expects this literal value to be compatible with the format that DBDATE specifies. Similarly, the database server interprets the date that you specify as the argument to the DATE() function to be in DBDATE format.
DATE Expressions in Database Objects

When an expression in a database object (including a check constraint, fragmentation expression, SPL routine, trigger, or UDR) contains a literal date value, the database server evaluates the expression using the setting that DBDATE (or other relevant environment variables) had when the database object was created (or was last modified). If DBDATE has been reset to a new value, the new value is ignored when the literal DATE is evaluated.

For example, suppose DBDATE is set to MDY2/ and a user creates a table with the following check constraint on the column orderdate:

orderdate < '06/25/98'

The date of the preceding expression is formatted according to the value of DBDATE when the constraint is defined. The check constraint expression is interpreted as orderdate < '06/25/98' regardless of the value of DBDATE during inserts or updates on the orderdate column. Suppose DBDATE is reset to MDY2/ when a user inserts the value '30/01/98' into the orderdate column. The date value inserted uses the date format MDY2/, whereas the check constraint expression uses the date format MDY2/.

See “Abbreviated Years and Expressions in Database Objects” on page 3-19 for a discussion of legacy objects from earlier versions of Informix database servers that are always evaluated according to the runtime environment. That section describes how to redefine objects so that dates are interpreted according to environment variable settings that were in effect when the object was defined (or when the object was last modified).

Important: The behavior of DBDATE for Dynamic Server and Extended Parallel Server is not backwards compatible.

DBDELMITER

The DBDELMITER environment variable specifies the field delimiter used with the dbexport utility and with the LOAD and UNLOAD statements.

```
>>> setenv DBDELMITER='delimeter'
```

`delimeter` is the field delimiter for unloaded data files.

The `delimeter` can be any single character, except those in the following list:

- Hexadecimal digits (0 through 9,a through f, A through F)
- Newline or CTRL-J
- The backslash ( \ ) symbol

The vertical bar ( | = ASCII 124) is the default. To change the field delimiter to a plus (+) symbol, for example, you can set DBDELMITER as follows:

```
setenv DBDELMITER '+'
```

DBEDIT

The DBEDIT environment variable specifies the text editor to use with SQL statements and command files in DB–Access. If DBEDIT is set, the specified text editor is invoked automatically. If DBEDIT is not, set you are prompted to specify a text editor as the default for the rest of the session.
To create, $INFORMIXDIR/msg have subdirectory full_path relative_path /SM590000/SM590000

The DB–Access utility displays the floating-point values of data types FLOAT, SMALLFLOAT, and DECIMAL(p) within a 14-character buffer. By default, DB–Access displays as many digits to the right of the decimal point as will fit into this character buffer. Therefore, the actual number of decimal digits that DB–Access displays depends on the size of the floating-point value.

To reduce the number of digits displayed to the right of the decimal point in floating-point values, set DBFLTMASK to the desired number of digits.

DBLANG

The DBLANG environment variable specifies the subdirectory of $INFORMIXDIR or the full pathname of the directory that contains the compiled message files that an IBM Informix product uses.

relative_path is a subdirectory of $INFORMIXDIR.
full_path is the pathname to the compiled message files.

By default, IBM Informix products put compiled messages in a locale-specific subdirectory of the $INFORMIXDIR/msg directory. These compiled message files have the file extension .iem. If you want to use a message directory other than $INFORMIXDIR/msg, where, for example, you can store message files that you create, you must perform the following steps:

To use a message directory other than $INFORMIXDIR/msg:
1. Use the `mkdir` command to create the appropriate directory for the message files.
   You can make this directory under the directory `$INFORMIXDIR` or `$/INFORMIXDIR/msg`, or you can make it under any other directory.

2. Set the owner and group of the new directory to `informix` and the access permission for this directory to 755.

3. Set the `DBLANG` environment variable to the new directory. If this is a subdirectory of `$INFORMIXDIR` or `$INFORMIXDIR/msg`, then you need only list the relative path to the new directory. Otherwise, you must specify the full pathname of the directory.

4. Copy the `.iem` files or the message files that you created to the new message directory that `$DBLANG` specifies.
   All the files in the message directory should have the owner and group `informix` and access permission 644.

IBM Informix products that use the default U.S. English locale search for message files in the following order:

1. In `$DBLANG`, if `DBLANG` is set to a full pathname
2. In `$INFORMIXDIR/msg/$DBLANG`, if `DBLANG` is set to a relative pathname
3. In `$INFORMIXDIR/$DBLANG`, if `DBLANG` is set to a relative pathname
4. In `$INFORMIXDIR/msg/en_us/0333`
5. In `$INFORMIXDIR/msg/en_us.8859-1`
6. In `$INFORMIXDIR/msg`
7. In `$INFORMIXDIR/msg/english`

For more information on search paths for messages, see the description of `DBLANG` in the IBM Informix GLS User’s Guide.

**DBMONEY**

The `DBMONEY` environment variable specifies the display format of values in columns of smallfloat, FLOAT, DECIMAL, or MONEY data types, and of complex data types derived from any of these data types.

```
$front-\$\back\$front-\back
```

- `$` is a currency symbol that precedes MONEY values in the default locale if no other `front` symbol is specified, or if `DBMONEY` is not set.
- `, or .` is a comma or period (the default) that separates the integral part from the fractional part of the FLOAT, DECIMAL, or MONEY value. Whichever symbol you do not specify becomes the thousands separator.
- `back` is a currency symbol that follows the MONEY value.
- `front` is a currency symbol that precedes the MONEY value.

The `back` symbol can be up to seven characters and can contain any character that the locale supports, except a digit, a comma (,), or a period (.) symbol. The `front` symbol can be up to seven characters and can contain any character that the locale supports.
supports except a digit, a comma (,), or a period (.) symbol. If you specify any character that is not a letter of the alphabet for front or back, you must enclose the front or back setting between single quotation ( ‘ ) marks.

When you display MONEY values, IBM Informix products use the DBMONEY setting to format the output. DBMONEY has no effect, however, on the internal format of data values that are stored in columns of the database.

If you do not set DBMONEY, then MONEY values for the default locale, U.S. English, are formatted with a dollar sign ($) that precedes the MONEY value, a period (.) that separates the integral from the fractional part of the MONEY value, and no back symbol. For example, 100.50 is formatted as $100.50.

Suppose you want to represent MONEY values as DM (deutsche mark) units, using the currency symbol DM and comma (,) as the decimal separator. Enter the following command to set the DBMONEY environment variable:

```
setenv DBMONEY DM,
```

Here DM is the front currency symbol that precedes the MONEY value, and a comma separates the integral from the fractional part of the MONEY value. As a result, the value 100.50 is displayed as DM100,50.

For more information about how DBMONEY formats MONEY values in nondefault locales, see the IBM Informix GLS User’s Guide.

**DBNLS (IDS)**

The DBNLS environment variable specifies whether automatic data type conversion is supported between NCHAR and NVARCHAR database columns and CHAR and VARCHAR variables (respectively) of client systems.

Global Language Support (GLS) does not require the DBNLS environment variable. But Dynamic Server databases continue to support the legacy behavior of DBNLS, which supports applications that manipulate tables with NCHAR or NVARCHAR columns.

```
setenv DBNLS 1
```

For UNIX systems that use the C shell, the following command line enables client applications such as DB-Access, IBM Informix SQL, IBM Informix 4GL, IBM Informix Dynamic 4GL, and embedded-SQL applications such as ESQL/C or ESQL/COBOL to convert automatically between CHAR and VARCHAR variables of the client application and NCHAR and NVARCHAR columns of the database:

```
setenv DBNLS 1
```

This setting also supports the automatic conversion of values retrieved from NCHAR columns into CHAR variables, and the conversion of NVARCHAR column values into VARCHAR variables.

Similarly, when DBNLS = 1, character strings stored as CHAR variables can be inserted into NCHAR columns, and character strings stored as VARCHAR variables can be inserted into NVARCHAR database columns.
To support these features, **DBNLS** must also be set to 1 on the client system. This setting also enables the client system to display dates, numbers, and currency values in formats specified on the client locale.

Conversely, each of the following command lines disables automatic conversion between CHAR and VARCHAR variables of the client application and NCHAR and NVARCHAR columns of the database and also prevents Dynamic Server from using the locale files of the client system:

```bash
s.getenv DBNLS

unsetenv DBNLS
```

On UNIX systems that use the C shell, either of these commands disables automatic conversion to and from NCHAR and NVARCHAR data values (by setting no value for **DBNLS**).

Another possible setting for **DBNLS** is 2. If you enter at the command line

```bash
setenv DBNLS 2
```

then automatic data type conversion between NCHAR and CHAR and between NVARCHAR and VARCHAR is supported (if the client system has **DBNLS** set to 1 or 2), but the database server can have a different locale from the client system.

### DBONPLOAD (IDS)

The **DBONPLOAD** environment variable specifies the name of the database that the **onload** utility of the High-Performance Loader (HPL) uses. If **DBONPLOAD** is set, **onload** uses the specified name as the name of the database; otherwise, the default name of the database is **onload**.

```bash
setenv DBONPLOAD dbname
```

**dbname** specifies the name of the database that the **onload** utility uses.

For example, to specify the name **load_db** as the name of the database, enter the following command:

```bash
setenv DBONPLOAD load_db
```

For more information, see the *IBM Informix High-Performance Loader User's Guide*.

### DBPATH

The **DBPATH** environment variable identifies database servers that contain databases. **DBPATH** can also specify a list of directories (in addition to the current directory) in which DB–Access looks for command scripts (**.sql** files).

The CONNECT DATABASE, START DATABASE, and DROP DATABASE statements use **DBPATH** to locate the database under two conditions:

- If the location of a database is not explicitly stated
- If the database cannot be located in the default server

The CREATE DATABASE statement does not use **DBPATH**.

To add a new **DBPATH** entry to existing entries, see “Modifying an Environment-Variable Setting” on page 3-6.
**full_pathname** is the full path, from root, of a directory where .sql files are stored.

**pathname** is the valid relative path of a directory where .sql files are stored.

**servername** is the name of an Informix database server where databases are stored. You cannot reference database files with a *servername*.

**DBPATH** can contain up to 16 entries. Each entry must be less than 128 characters. In addition, the maximum length of **DBPATH** depends on the hardware platform on which you set **DBPATH**.

When you access a database with the CONNECT, DATABASE, START DATABASE, or DROP DATABASE statement, the search for the database is done first in the directory or database server specified in the statement. If no database server is specified, the default database server that was specified by the **INFORMIXSERVER** environment variable is used.

If the database is not located during the initial search, and if **DBPATH** is set, the database servers and directories in **DBPATH** are searched for in the specified database. These entries are searched in the same order in which they are listed in the **DBPATH** setting.

**Using DBPATH with DB-Access**

If you use DB-Access and select the Choose option from the SQL menu without having already selected a database, you see a list of all the .sql files in the directories listed in your **DBPATH**. After you select a database, the **DBPATH** is not used to find the .sql files. Only the .sql files in the current working directory are displayed.

**Searching Local Directories**

Use a pathname without a database server name to search for .sql scripts on your local computer. In the following example, the **DBPATH** setting causes DB-Access to search for the database files in your current directory and then in the Joachim and Sonja directories on the local computer:

```
setenv DBPATH /usr/joachim:/usr/sonja
```

As the previous example shows, if the pathname specifies a directory name but not a database server name, the directory is sought on the computer that runs the default database server that the **INFORMIXSERVER** specifies; see "**INFORMIXSERVER** on page 3-50." For instance, with the previous example, if **INFORMIXSERVER** is set to quality, the **DBPATH** value is interpreted, as the following example shows, where the double slash precedes the database server name:

```
setenv DBPATH //quality/usr/joachim://quality/usr/sonja
```

**Searching Networked Computers for Databases**

If you use more than one database server, you can set **DBPATH** explicitly to contain the database server and directory names that you want to search for databases. For example, if **INFORMIXSERVER** is set to quality, but you also want to search the marketing database server for /usr/joachim, set **DBPATH** as the following example shows:
setenv DBPATH //marketing/usr/joachim:/usr/sonja

**Specifying a Servername**
You can set DBPATH to contain only database server names. This feature allows you to locate only databases; you cannot use it to locate command files.

The database administrator must include each database server mentioned by DBPATH in the $INFORMIXDIR/etc/sqlhosts file. For information on communication-configuration files and dbservername, see your IBM Informix Administrator’s Guide and the IBM Informix Administrator’s Reference.

For example, if INFORMIXSERVER is set to quality, you can search for a database first on the quality database server and then on the marketing database server by setting DBPATH, as the following example shows:

```bash
setenv DBPATH //marketing
```

If you use DB–Access in this example, the names of all the databases on the quality and marketing database servers are displayed with the Select option of the DATABASE menu.

**DBPRINT**
The DBPRINT environment variable specifies the default printing program.

```bash
setenv DBPRINT program
```

*program* is any command, shell script, or UNIX utility that produces standard ASCII output.

If you do not set DBPRINT, the default program is found in one of two places:
- For most BSD UNIX systems, the default program is lpr.
- For UNIX System V, the default program is usually lp.

Enter the following command to set the DBPRINT environment variable to specify myprint as the print program:

```bash
setenv DBPRINT myprint
```

**DBREMTEDCMD (UNIX)**
Set the DBREMTEDCMD environment variable to override the default remote shell to perform remote tape operations with the database server. You can set dbremotecmd to a simple command or to a full pathname.

```bash
setenv DBREMTEDCMD command pathname
```

*command* is a command to override the default remote shell.

*pathname* is a pathname to override the default remote shell.

If you do not specify the full pathname, the database server searches your PATH for the specified command. It is highly recommended that you use the full pathname syntax on interactive UNIX platforms to avoid problems with similarly named programs in other directories and possible confusion with the restricted shell (/usr/bin/rsh).
The following command sets DBREMOTECMD for a simple command name:
setenv DBREMOTECMD rcmd

The next command to set DBREMOTECMD specifies a full pathname:
setenv DBREMOTECMD /usr/bin/remsh

For more information on DBREMOTECMD, see the discussion in your IBM Informix Backup and Restore Guide about how to use remote tape devices with your database server for archives, restores, and logical-log backups.

**DBSPACETEMP**

The DBSPACETEMP environment variable specifies the dbspaces in which temporary tables are built.

You can list dbspaces, separated by colon (:) or comma (,) symbols to spread temporary space across any number of disks.

```
setenv DBSPACETEMP temp_dbspace
```

`temp_dbspace` is the name of a valid existing temporary dbspace.

DBSPACETEMP overrides any default dbspaces that the DBSPACETEMP parameter specifies in the configuration file of the database server. For UPDATE STATISTICS, DBSPACETEMP is used only when you specify the option HIGH. You might have better performance if the list of dbspaces in DBSPACETEMP is composed of chunks that are allocated as raw UNIX devices.

For example, the following command to set the DBSPACETEMP environment variable specifies three dbspaces for temporary tables:

```
setenv DBSPACETEMP sorttmp1:sorttmp2:sorttmp3
```

Separate the dbspace entries with either colons or commas. The number of dbspaces is limited by the maximum size of the environment variable, as defined by your operating system. Your database server does not create a dbspace specified by the environment variable if the dbspace does not exist.

The two classes of temporary tables are *explicit* temporary tables that the user creates and *implicit* temporary tables that the database server creates. Use DBSPACETEMP to specify the dbspaces for both types of temporary tables.

If you create an explicit temporary table with the CREATE TEMP TABLE statement and do not specify a dbspace for the table either in the IN dbspace clause or in the FRAGMENT BY clause, the database server uses the settings in DBSPACETEMP to determine where to create the table.

If you create an explicit temporary table with the SELECT INTO TEMP statement, the database server uses the settings in DBSPACETEMP to determine where to create the table.

If DBSPACETEMP is set, and the dbspaces that it lists include both logging and non-logging dbspaces, the database server stores temporary tables that implicitly or explicitly support transaction logging in a logged dbspace, and non-logging temporary tables in a non-logging dbspace.
The database server creates implicit temporary tables for its own use while executing join operations, SELECT statements with the GROUP BY clause, SELECT statements with the ORDER BY clause, and index builds.

When it creates explicit or implicit temporary tables, the database server uses disk space for writing the temporary data. If there are conflicts among settings or statement specifications for the location of a temporary table, these conflicts are resolved in this descending (highest to lowest) order of precedence:

1. What the IN or FRAGMENT BY clause of a DDL or DML statement specifies
2. For Extended Parallel Server, what a SET TEMP TABLE_SPACE statement specifies
3. On UNIX platforms, the operating-system directory or directories that the environment variable PSORT_DBTEMP specifies, if this is set
4. The dbspace or dbspaces that the environment variable DBSPACETEMP specifies, if this is set
5. The dbspace or dbspaces that the ONCONFIG parameter DBSPACETEMP specifies.
6. The dbspace or dbspaces that the ONCONFIG parameter TABLESPACE specifies.
7. The operating-system file space in /tmp (UNIX) or %temp% (Windows)
8. For Extended Parallel Server, in a non-critical space, if none of the above are specified
9. For Dynamic Server, in the space where the database was created, if none of the above are specified

Important: If the DBSPACETEMP environment variable is set to an invalid value, the database server defaults to the root dbspace for explicit temporary tables and to /tmp for implicit temporary tables, not to the DBSPACETEMP configuration parameter. In this situation, the database server might fill /tmp to the limit and eventually bring down the database server or kill the file system.

**DBTEMP (IDS)**

The DBTEMP environment variable is used by DB-Access and IBM Informix Enterprise Gateway products as well as by Dynamic Server and by earlier database servers. DBTEMP resembles DBSPACETEMP, specifying the directory in which to place temporary files and temporary tables.

```bash
setenv DBTEMP pathname
```

*pathname* is the full pathname of the directory for temporary files and tables.

For DB-Access to work correctly on Windows platforms, DBTEMP should be set to $INFORMIXDIR/infxtmp.

The following example sets DBTEMP to the pathname *usr/magda/mytemp* for UNIX systems that use the C shell:

```
setenv DBTEMP usr/magda/mytemp
```

Important: DBTEMP can point to an NFS-mounted directory only if the vendor of that NFS device is certified by IBM.
If DBTEMP is not set, the database server creates temporary files in the /tmp directory and temporary tables in the DBSPACETEMP directory. See "DBSPACETEMP" on page 3-29 for the default if DBSPACETEMP is not set.

Similarly, if you do not set DBTEMP on the client system, temporary files (such as those created for scroll cursors) are created in the /tmp directory.

You might experience unexpected behavior or failure in operations on values of large or complex data types, such as BYTE or ROW, if DBTEMP is not set.

**DBTIME**

The DBTIME environment variable specifies a formatting mask for the display and data-entry format of DATETIME values. The DBTIME environment variable is useful in contexts where the DATETIME data values to be formatted by DBTIME have the same precision as the specified DBTIME setting. You might encounter unexpected or invalid display formats for DATETIME values that are declared with a different DATETIME qualifier.

```
setenv DBTIME 'literal %k min precision special'
```

- **literal** is a literal white space or any printable character.
- **min** is a literal integer, setting the minimum number of characters in the substring for the value that special specifies.
- **precision** is the number of digits for the value of any time unit, or the maximum number of characters in the name of a month.
- **special** is one of the placeholder characters that are listed following.

These terms and symbols are described in the pages that follow.

This quoted string can include literal characters as well as placeholders for the values of individual time units and other elements of a DATETIME value. DBTIME takes effect only when you call certain IBM Informix ESQL/C DATETIME routines. (For details, see the IBM Informix ESQL/C Programmer's Manual.) If DBTIME is not set, the behavior of these routines is undefined, and "YYYY-MM-DD hh:mm:ss.ffffff" is the default display and input format for DATETIME YEAR TO FRACTION(5) literal values in the default locale.

The percentage ( % ) symbol gives special significance to the special placeholder symbol that follows. Without a preceding % symbol, any character within the formatting mask is interpreted as a literal character, even if it is the same character as one of the placeholder characters in the following list. Note also that the special placeholder symbols are case sensitive.

The following characters within a DBTIME format string are placeholders for time units (or for other features) within a DATETIME value.

- **%b** is replaced by the abbreviated month name.
- **%B** is replaced by the full month name.
- **%d** is replaced by the day of the month as a decimal number [01,31].
%FH  is replaced by a fraction of a second with a scale that the integer n specifies. The default value of n is 2; the range of n is 0 ≤ n ≤ 5.

%H  is replaced by the hour (24-hour clock).

%I  is replaced by the hour (12-hour clock).

%M  is replaced by the minute as a decimal number [00,59].

%m  is replaced by the month as a decimal number [01,12].

%p  is replaced by A.M. or P.M. (or the equivalent in the locale file).

%S  is replaced by the second as a decimal number [00,59].

%y  is replaced by the year as a four-digit decimal number.

%Y  is replaced by the year as a four-digit decimal number. User must enter a four-digit value.

%%%  is replaced by % (to allow % in the format string).

For example, consider this display format for DATETIME YEAR TO SECOND:

Mar 21, 2001 at 16 h 30 m 28 s

If the user enters a two-digit year value, this value is expanded to 4 digits according to the DBCENTURY environment variable setting. If DBCENTURY is not set, then the string 19 is used by default for the first two digits.

Set DBTIME as the following command line (for the C shell) shows:

```
setenv DBTIME '\%b \%d, \%Y at \%H \%M \%S s'
```

The default DBTIME produces the following ANSI SQL string format:

```
2001-03-21 16:30:28
```

You can set the default DBTIME as the following example shows:

```
setenv DBTIME '\%Y-\%m-\%d \%H:\%M:\%S'
```

An optional field width and precision specification (w.p) can immediately follow the percent (%) character. It is interpreted as follows:

- **w**  Specifies the minimum field width. The value is right-justified with blank spaces on the left.
- **-w**  Specifies the minimum field width. The value is left-justified with blank spaces on the right.
- **0w**  Specifies the minimum field width. The value is right-justified and padded with zeros on the left.

- **p**  Specifies the precision of d, H, I, m, M, S, y, and Y time unit values, or the maximum number of characters in b and B month names.

The following limitations apply to field-width and precision specifications:

- If the data value supplies fewer digits than precision specifies, the value is padded with leading zeros.
- If a data value supplies more characters than precision specifies, excess characters are truncated from the right.
- If no field width or precision is specified for d, H, I, m, M, S, or y placeholders, 0.2 is the default, or 0.4 for the Y placeholder.
A precision specification is significant only when converting a DATETIME value to an ASCII string, but not vice versa.

The $F$ placeholder does not support this field-width and precision syntax.

Like DBDATE, GL_DATE, or GL_DATETIME, the DBTIME setting controls only the character-string representation of data values; it cannot change the internal storage format of the DATETIME column. (For information about formatting DATE values, see the discussion of DBDATE on page 3-20)

In East Asian locales that support era-based dates, DBTIME can also specify Japanese or Taiwanese eras. See IBM Informix GLS User's Guide for details of additional placeholder symbols for setting DBTIME to display era-based DATETIME values, and for descriptions of the GL_DATETIME and GL_DATE environment variables.

**DBUPSPACE**

The DBUPSPACE environment variable lets you specify and constrain the amount of system disk space that the UPDATE STATISTICS statement can use when trying to simultaneously construct multiple column distributions.

```
setenv DBUPSPACE max : default : option
```

- **max** is a positive integer, specifying the maximum disk space (in kilobytes) to allocate for sorting in UPDATE STATISTICS operations.
- **default** is a positive integer, specifying the maximum amount of memory (from 4 to 50 megabytes) to allocate without using PDQ.
- **option** An unsigned integer:
  - 1: Do not use any indexes for sorting. Print the entire plan for update statistics in sqexplain.out.
  - 2: Do not use any indexes for sorting. Do not print the plan for update statistics.
  - 3 or greater: Use available indexes for sorting. Print the entire plan for update statistics in sqexplain.out.

For example, to set DBUPSPACE to 2,500 kilobytes of disk space and 1 megabyte of memory, enter this command:

```
setenv DBUPSPACE 2500:1
```

After you set this value, the database server can use no more than 2,500 kilobytes of disk space during the execution of an UPDATE STATISTICS statement. If a table requires 5 megabytes of disk space for sorting, then UPDATE STATISTICS accomplishes the task in two passes; the distributions for one half of the columns are constructed with each pass.

If you do not set DBUPSPACE, the default is one megabyte (1,024 kilobytes) for max, and 15 megabytes for default. If you attempt to set DBUPSPACE to any value less than 1,024 kilobytes, it is automatically set to 1,024 kilobytes, but no error message is returned. If this value is not large enough to allow more than one distribution to be constructed at a time, at least one distribution is done, even if the amount of disk space required to do this is more than what DBUPSPACE specifies.
DEFAULT.Attach

The DEFAULT.Attach environment variable supports the legacy behavior of Version 7.x of Dynamic Server, which required that only nonfragmented B-tree indexes on nonfragmented tables can be attached.

```
$ setenv DEFAULT_ATTACH 1
```

If DEFAULT.Attach is set to 1, then all other indexes that you create, including R-trees and UDR functional indexes, must be detached, unless you specify the IN TABLE keywords as the Storage clause of the CREATE INDEX statement. (An attached index is one that has the same distribution scheme as the table on which it is built. Any index that does not is a detached index.)

If DEFAULT.Attach is not set, then any CREATE INDEX statement that does not specify IN TABLE as its Storage clause creates detached indexes by default. This release of Dynamic Server can support attached indexes that were created by Version 7.x of Dynamic Server.

**Important**: Future releases of Informix database servers might not continue to support DEFAULT.Attach. Developing new applications that depend on this deprecated feature is not recommended.

DELIMIDENT

The DELIMIDENT environment variable specifies that strings enclosed between double quotation (" ) marks are delimited database identifiers. DELIMIDENT is also supported on client systems, where it can be set to y, to n, or to no setting.

- y specifies that client applications must use single quote (’ ) symbols to delimit character strings, and must use double quote (" ) symbols only around delimited SQL identifiers, which can support a larger character set than is valid in undelimited identifiers. Letters within delimited strings or delimited identifiers are case-sensitive. This is the default value for OLE DB and .NET.
- n specifies that client applications can use double quote (" ) or single quote (’ ) symbols to delimit character strings, but not to delimit SQL identifiers. If the database server encounters a string delimited by double or single quote symbols in a context where an SQL identifier is required, it issues an error. An owner name that qualifies an SQL identifier can be delimited by single quote (’ ) symbols. You must use a pair of the same quote symbols to delimit a character string.

This is the default value for ESQL/C, JDBC, and ODBC. APIs that have ESQL/C as an underlying layer, such as Informix 4GL, the DataBlade API (LIBDMI), and the C++ API, behave as ESQL/C, and use ‘n’ as the default if no value for DELIMIDENT is specified on the client system.

- Specifying DELIMIDENT with no value on the client system requires client applications to use the DELIMIDENT setting that is the default for their application programming interface (API).

```
$ setenv DELIMIDENT
```

No value is required; DELIMIDENT takes effect if it exists, and it remains in effect while it is on the list of environment variables. Removing DELIMIDENT when it is set at the server level requires restarting the server.
Delimited identifiers can include white space (such as the phrase "Vitamin E") or can be identical to SQL keywords, (such as "TABLE" or "USAGE"). You can also use them to declare database identifiers that contain characters outside the default character set for SQL identifiers (such as "Column #6"). In the default locale, this set consists of letters, digits, and the underscore ( _ ) symbol.

Even if DELIMIDENT is set, you can use single quote ( ’ ) symbols to delimit authorization identifiers as the owner name component of a database object name, as in the following example:

```
RENAME COLUMN 'Owner'.table2.collum3 T0 column3
```

This example is an exception to the general rule that when DELIMIDENT is set, the SQL parser interprets character strings delimited by single quotes as string literals, and interprets character strings delimited by double quotes ( " ) as SQL identifiers.

*Database identifiers* (also called *SQL identifiers*) are names for database objects, such as tables and columns. *Storage identifiers* are names for storage objects, such as dbspaces, blobspaces, and ssbspaces (smart blob spaces). You cannot use **DELIMIDENT** to declare storage identifiers that contain characters outside the default SQL character set.

Delimited identifiers are case sensitive. To use delimited identifiers, applications in ESQL/C must set **DELIMIDENT** at compile time and at runtime.

**Warning:** If **DELIMIDENT** is not already set, you should be aware that setting it can cause the failure of existing .sql scripts or client applications that use double ( " ) quotation marks in contexts other than delimiting SQL identifiers, such as delimiters of string literals. You must use single ( ’ ) rather than double quotation marks for delimited constructs that are not SQL identifiers if **DELIMIDENT** is set.

On UNIX systems that use the C shell and on which **DELIMIDENT** has been set, you can disable this feature (which causes anything between double quotes to be interpreted as an SQL identifier) by the command:

```
unsetenv  DELIMIDENT
```

**ENVIGNORE (UNIX)**

The **ENVIGNORE** environment variable can deactivate specified environment variable settings in the common (shared) and private environment-configuration files, **informix.rc** and **.informix** respectively.

```
$setenv ENVIGNORE variable
```

*variable* is the name of an environment variable to be deactivated.

Use colon ( : ) symbols between consecutive *variable* names. For example, to ignore the **DBPATH** and **DBMONEY** entries in the environment-configuration files, enter the following command:

```
setenv ENVIGNORE DBPATH:DBMONEY
```

The common environment-configuration file is stored in **$INFORMIXDIR/etc/informix.rc**.
The private environment-configuration file is stored in the user’s home directory as .informix.

For information on creating or modifying an environment-configuration file, see “Setting Environment Variables in a Configuration File” on page 3-4.

ENVIGNORE itself cannot be set in an environment-configuration file.

**FET_BUF_SIZE**

The **FET_BUF_SIZE** environment variable can override the default setting for the size of the fetch buffer for all data types except BYTE and TEXT values.

```
setenv FET_BUF_SIZE size
```

`size` is a positive integer that is larger than the default buffer size, but no greater than 32,767, specifying the size (in bytes) of the fetch buffer that holds data retrieved by a query.

For example, to set a buffer size to 5,000 bytes on a UNIX system that uses the C shell, set **FET_BUF_SIZE** by entering the following command:

```
setenv FET_BUF_SIZE 5000
```

When **FET_BUF_SIZE** is set to a valid value, the new value overrides the default value (or any previously set value of **FET_BUF_SIZE**). The default setting for the fetch buffer is dependent on row size.

The processing of BYTE and TEXT values is not affected by **FET_BUF_SIZE**.

No error is raised if **FET_BUF_SIZE** is set to a value that is less than the default size or that is out of the range of SMALLINT values. In these cases, however, the invalid fetch buffer size is ignored, and the default size is in effect.

A valid **FET_BUF_SIZE** setting is in effect for the local database server as well as for any remote database server from which you retrieve rows through a distributed query in which the local server is the coordinator and the remote database is subordinate. The greater the size of the buffer, the more rows can be returned, and the less frequently the client application must wait while the database server returns rows. A large buffer can improve performance by reducing the overhead of filling the client-side buffer.

**GLOBAL_DETACH_INFORM (XPS)**

All indexes in IBM Informix Extended Parallel Server are detached. An XPS index is *locally detached* when every index fragment resides on the same coserver as its associated data fragment and XPS index is *globally detached* if it is fragmented, with index items and their associated data rows residing on different coservers.

You should avoid using globally detached indexes because they are inherently less efficient than locally detached indexes.

The **GLOBAL_DETACH_INFORM** environment variable triggers an alarm if a globally detached index is created. The alarm has a severity of 3 (Attention), a Class ID of 10 (Performance Improvement Possible) and a Tag ID of 1 (Globally Detached Index Built).
To enable this alarm, set `GLOBAL_DETACH_INFORM` to any value before starting the server.

```
setenv GLOBAL_DETACH_INFORM n
```

Alternatively, you can turn this variable on or off with the `onutil SET` command, as in the following example:

```
% onutil
1> SET GLOBAL_DETACH_INFORM 1;
Dynamic Configuration completed successfully
```

Because `GLOBAL_DETACH_INFORM` is an environment variable and not a configuration parameter, however, it cannot be made persistent with the `onutil` command. Add the variable to an environment-configuration file to avoid setting it each time the server is restarted.

**IBM_XPS_PARAMS (XPS)**

By default, the `CURRENT` and `TODAY` functions return values from the system clock-calendar, based on the location of the server. Use the `IBM_XPS_PARAMS` environment variable to specify a non-default time zone, as an offset from Greenwich Mean Time (GMT), for values returned by `CURRENT` and `TODAY`.

To specify an offset from GMT for the built-in `CURRENT` and `TODAY` functions, set `IBM_XPS_PARAMS` on the client system to the desired value before you start the client application, using the following syntax.

```
setenv IBM_XPS_PARAMS 'CLIENT_TZ = [+-] hours:minutes'
```

*hours* is an integer in the range from 0 to 13 inclusive, specifying the absolute number of hours in the offset from GMT.

*minutes* is a 2-digit integer in the range from 00 to 59 inclusive, specifying any additional minutes in the offset from GMT.

A minus (-) sign before the *hours* specifies a time zone east of GMT, and a positive (+) sign, which is the default, specifies a time zone west of GMT.

This example specifies that `CURRENT` and `TODAY` return GMT values:

```
% setenv IBM_XPS_PARAMS 'CLIENT_TZ = 00:00'
```

The next example specifies the Atlantic time zone of eastern Canada:

```
% setenv IBM_XPS_PARAMS 'CLIENT_TZ = +4:00'
```

The `onstat -g ses` command can display the current offset from GMT.

The `SET ENVIRONMENT` `CLIENT_TZ` statement of SQL can override the `IBM_XPS_PARAMS` setting, but the default scope of this environment variable is all sessions, rather than only the session in which the `SET ENVIRONMENT CLIENT_TZ` statement is issued. Reset the GMT offset with the SQL statement for the current session if your application requires a different time zone.
IFMX_CART_ALRM (XPS)

The IFMX_CART_ALRM environment variable triggers an alarm if a query executes a Cartesian join. The alarm has a severity of 3 (Attention), a Class ID of 10 (Performance Improvement Possible) and a Tag ID of 2 (Cartesian Join Processing). The alarm message indicates the ID of the session executing the Cartesian join.

To enable this alarm, set IFMX_CART_ALRM to any value before starting the database server.

```
setenv IFMX_CART_ALRM n
```

Alternatively, you can turn this variable on or off with the onutil SET command:

```
% onutil
1> SET IFMX_CART_ALRM 1;
Dynamic Configuration completed successfully
```

Because IFMX_CART_ALRM is an environment variable and not a configuration parameter, it cannot be made persistent by the onutil command. Add the variable to an environment-configuration file to avoid setting it each time the server is restarted.

IFMX_HISTORY_SIZE (XPS)

The IFMX_HISTORY_SIZE environment variable determines the number of SQL commands that are logged in the DB-Access command history.

```
setenv IFMX_HISTORY_SIZE value
```

value the number of commands stored in the DB-Access history

The default value is 10. The maximum is 100. If a value greater or lower is specified, the default value is used. For more information on using the DB-Access history command, see the IBM Informix DB-Access User's Guide.

IFMX_OPT_FACT_TABS (XPS)

The IFMX_OPT_FACT_TABS environment variable specifies a list of fact tables that should be used in push-down hash joins whenever possible.

```
setenv IFMX_OPT_FACT_TABS fact_table database:owner.
```

database is name of the database.

fact_table is name of the fact table.

owner is name of the table owner.

If you do not specify a database name or owner, the fact table can be in any database or belong to any owner.
The environment variable lists fact tables for which you want to encourage the optimizer to choose push-down hash-join plans. If you do not specify the database name or owner, the table can be in any database or belong to any owner.

When this environment variable is set, push-down hash-join restrictions for the specified fact tables are relaxed to allow the optimizer to choose a push-down plan even when the fact table is not larger than the dimension table or when the dimension-table join columns are not unique.

You can use IFMX_OPT_FACT_TABS alone to increase the possibility of push-down hash joins. You can also use it in conjunction with the IFMX_OPT_NON_DIM_TABS environment variable to fine tune the use of push-down hash joins.

**IFMX_OPT_NON_DIM_TABS (XPS)**

The IFMX_OPT_NON_DIM_TABS environment variable specifies a list of dimension tables that cannot be used in push-down hash-join query plans. If the optimizer detects a fact-dimension table query that joins one of these dimension tables, it does not choose a push-down hash-join plan.

You can use the IFMX_OPT_NON_DIM_TABS environment variable alone to decrease the possibility of push-down hash joins. You can also use it in conjunction with the IFMX_OPT_FACT_TABS environment variable to fine tune the use of push-down hash joins.

**IFX_DEF_TABLE_LOCKMODE (IDS)**

The IFX_DEF_TABLE_LOCKMODE environment variable can specify the default lock mode for database tables that are subsequently created without explicitly specifying the LOCKMODE PAGE or LOCKMODE ROW keywords. This feature is convenient if you need to create several tables of the same lock mode. UNIX systems that use the C shell support the following syntax:

The default lock mode is page-level granularity. This value disables the LAST COMMITTED feature of COMMITTED READ.
ROW The default lock mode is row-level granularity.

Similar functionality is available by setting the DEF_TABLE_LOCKMODE parameter of the ONCONFIG file to PAGE or ROW. When a table is created or modified, any conflicting lock mode specifications are resolved according to the following descending (highest to lowest) order of precedence:

1. Explicit LOCKMODE specification of CREATE TABLE or ALTER TABLE
2. IFX_DEF_TABLE_LOCKMODE environment variable setting
3. DEF_TABLE_LOCKMODE parameter setting in the ONCONFIG file
4. The system default lock mode (= page mode)

To make the DEF_TABLE_LOCKMODE setting the default mode (or to restore the system default if DEF_TABLE_LOCKMODE is not set) use the command:

```
unsetenv IFX_DEF_TABLE_LOCKMODE
```

If IFX_DEF_TABLE_LOCKMODE is set in the environment of the database server before running oninit, then its scope is all sessions of the database server (just as if DEF_TABLE_LOCKMODE were set in the ONCONFIG file). If IFX_DEF_TABLE_LOCKMODE is set in the shell, or in the $HOME/informix or $INFORMIXDIR/etc/informix.rc files, then the scope is restricted to the current session (if you set it in the shell) or to the individual user.

**Important:** This has no effect on existing tables. If you specify ROW as the lock mode, the database will use this to restore, recover, or copy data. For tables that were created in PAGE mode, this might cause lock-table overflow or performance degradation.

**IFX_DIRECTIVES**

The IFX_DIRECTIVES environment variable setting determines whether the optimizer allows query optimization directives from within a query. The IFX_DIRECTIVES environment variable is set on the client.

You can specify either ON and OFF or 1 and 0 to set the environment variable.

```
setenv IFX_DIRECTIVES 0
```

1 Optimizer directives accepted
0 Optimizer directives not accepted

The setting of the IFX_DIRECTIVES environment variable overrides the value of the DIRECTIVES configuration parameter that is set for the database server. If the IFX_DIRECTIVES environment variable is not set, however, then all client sessions will inherit the database server configuration for directives that the ONCONFIG parameter DIRECTIVES determines. The default setting for the IFX_DIRECTIVES environment variable is ON.

For more information about the DIRECTIVES parameter, see the *IBM Informix Administrator’s Reference*. For more information on the performance impact of directives, see your *IBM Informix Performance Guide*. 

3-40 IBM Informix Guide to SQL: Reference
**IFX_EXTDIRECTIVES**

The IFX_EXTDIRECTIVES environment variable specifies whether the query optimizer allows external query optimization directives from the sysdirectives system catalog table to be applied to queries in existing applications. The IFX_EXTDIRECTIVES environment variable is set on the client.

You can specify either ON and OFF or 1 and 0 to set the environment variable.

```
setenv IFX_DIRECTIVES 1 0
```

<table>
<thead>
<tr>
<th>IFX_EXTDIRECTIVES</th>
<th>EXT_DIRECTIVES = 0</th>
<th>EXT_DIRECTIVES = 1</th>
<th>EXT_DIRECTIVES = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>No setting</td>
<td>OFF</td>
<td>OFF</td>
<td>ON</td>
</tr>
<tr>
<td>0 = OFF</td>
<td>OFF</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>1 = ON</td>
<td>OFF</td>
<td>ON</td>
<td>ON</td>
</tr>
</tbody>
</table>

The database server interprets any EXT_DIRECTIVES setting besides 1 or 2 (or no setting) as equivalent to OFF, disabling support for external directives. Any value of IFX_EXTDIRECTIVES other than 1 has the same effect for the client.

For information on how to define external optimizer directives, see the description of the SAVE EXTERNAL DIRECTIVES statement of SQL in the IBM Informix Guide to SQL: Syntax. For more information about the EXT_DIRECTIVES configuration parameter, see the IBM Informix Administrator’s Reference. For more information on the performance impact of directives, see your IBM Informix Performance Guide.

**IFX_LONGID**

The IFX_LONGID environment variable setting and the version number of the client application determine whether a given client application is capable of handling long identifiers. (Older versions of Informix databases restricted SQL identifiers to 18 or fewer bytes; long identifiers can have up to 128 bytes when IFX_LONGID is set.) Valid IFX_LONGID values are 1 and 0.

```
setenv IFX_LONGID 1 0
```
1 Client supports long identifiers.
0 Client cannot support long identifiers.

When IFX_LONGID is set to zero, applications display only the first 18 bytes of long identifiers, without indicating (by +) that truncation has occurred.

If IFX_LONGID is unset or is set to a value other than 1 or 0, the determination is based on the internal version of the client application. If the (server-based) version is not less than 9.0304, or is in the (CSDK-based) range 2.90 ≤ version < 4.0, the client is considered capable of handling long identifiers. Otherwise, the client application is considered incapable.

The IFX_LONGID setting overrides the internal version of the client application. If the client cannot handle long identifiers despite a newer version number, set IFX_LONGID to 0. If the client version can handle long identifiers despite an older version number, set IFX_LONGID to 1.

If you set IFX_LONGID on the client, the setting affects only that client. If you bring up the database server with IFX_LONGID set, all client applications use that setting by default. If IFX_LONGID is set to different values on the client and on the database server, however, the client setting takes precedence.

**Important:** ESQL executables that have been built with the -static option using the libos.a library version that does not support long identifiers cannot use the IFX_LONGID environment variable. You must recompile such applications with the new libos.a library that includes support for long identifiers. Executables that use shared libraries (no -static option) can use IFX_LONGID without recompilation provided that they use the new libfos.so that provides support for long identifiers. For details, see your ESQL product publication.

**IFX_NETBUF_PVTPOOL_SIZE (UNIX)**

The IFX_NETBUF_PVTPOOL_SIZE environment variable specifies the maximum size of the free (unused) private network buffer pool for each database server session.

```
        setenv IFX_NETBUF_PVTPOOL_SIZE count
```

*count* is an integer specifying the number of units (buffers) in the pool.

The default size is 1 buffer. If IFX_NETBUF_PVTPOOL_SIZE is set to 0, then each session obtains buffers from the free global network buffer pool. You must specify the value in decimal form.

**IFX_NETBUF_SIZE**

The IFX_NETBUF_SIZE environment variable lets you configure the network buffers to the optimum size. It specifies the size of all network buffers in the free (unused) global pool and the private network buffer pool for each database server session.

```
        setenv IFX_NETBUF_SIZE size
```

*size* is the integer size (in bytes) for one network buffer.
The default size is 4 kilobytes (4,096 bytes). The maximum size is 64 kilobytes (65,536 bytes) and the minimum size is 512 bytes. You can specify the value in hexadecimal or decimal form.

**Tip:** You cannot set a different size for each session.

### IFX_NO_LRU_TUNING

The IFX_NO_LRU_TUNING environment variable lets you configure the network buffers to the optimum size. It specifies the size of all network buffers in the free (unused) global pool and the private network buffer pool for each database server session.

```bash
> setenv IFX_NETBUF_SIZE size
```

`size` is the integer size (in bytes) for one network buffer.

The default size is 4 kilobytes (4,096 bytes). The maximum size is 64 kilobytes (65,536 bytes) and the minimum size is 512 bytes. You can specify the value in hexadecimal or decimal form.

**Tip:** You cannot set a different size for each session.

### IFX_NO_TIMELIMIT_WARNING

Trial or evaluation versions of IBM Informix software products, which cease to function when some time limit has elapsed since the software was installed, by default issue warning messages that tell users when the license will expire. If you set the IFX_NO_TIMELIMIT_WARNING environment variable, however, the time-limited software does not issue these warning messages.

```bash
> setenv IFX_NO_TIMELIMIT_WARNING
```

For users who dislike viewing warning messages, this feature is an alternative to redirecting the error output. Setting IFX_NO_TIMELIMIT_WARNING has no effect, however, on when a time-limited license expires; the software ceases to function at the same point in time when it would if this environment variable had not been set. If you do set IFX_NO_TIMELIMIT_WARNING, users will not see potentially annoying warnings about the impending license expiration, but some users might be annoyed at you when the database server (or whatever software has a time-limited license) ceases to function without any warning.

### IFX_NODBPROC

The IFX_NODBPROC environment variable lets you prevent the database server from running the sysdbopen( ) or sysdclose( ) procedure. These procedures cannot be run if this environment variable is set to any value.

```bash
> setenv IFX_NODBPROC string
```

`string` Any value prevents the database server from running sysdbopen( ) or sysdclose( ).
**IFX_ONTAPE_FILE_PREFIX**

When TAPEDEV and LTAPEDEV specify directories, use the **IFX_ONTAPE_FILE_PREFIX** environment variable to specify a prefix for backup file names that replaces the `hostname_servernum` format. If no value is set, file names are `hostname_servernum_Ln` for levels and `hostname_servernum_Lognnnnnnnnn` for log files.

If you set the value of **IFX_ONTAPE_FILE_PREFIX** to `My_Backup`, the backup file names have the following names:

- My_Backup_L0
- My_Backup_L1
- My_Backup_L2
- My_Backup_Log0000000001
- My_Backup_Log0000000002

```bash
setenv IFX_ONTAPE_FILE_PREFIX string
```

`string`  The prefix to use for the names of backup files.

**IFX_PAD_VARCHAR (IDS)**

The **IFX_PAD_VARCHAR** environment variable setting controls how the database server sends and receives VARCHAR and NVARCHAR data values. Valid **IFX_PAD_VARCHAR** values are 1 and 0.

```bash
setenv IFX_PAD_VARCHAR 1 0
```

- 1  Transmit the entire structure, up to the declared max size.
- 0  Transmit only the portion of the structure containing data.

For example, to send the string "ABC" from a column declared as NVARCHAR(255) when **IFX_PAD_VARCHAR** is set to 0 would send 3 bytes.

If the setting were 1 in the previous example, however, the number of bytes sent would be 255 bytes.

The effect **IFX_PAD_VARCHAR** is context-sensitive. In a low-bandwidth network, a setting of 0 might improve performance by reducing the total volume of transmitted data. But in a high-bandwidth network, a setting of 1 might improve performance, if the CPU time required to process variable-length packets were greater than the time required to send the entire character stream. In cross-server distributed operations, this setting has no effect, and padding characters are dropped from VARCHAR or NVARCHAR values that are passed between database servers.

**IFX_UPDDESC (IDS)**

You must set the **IFX_UPDDESC** environment variable at execution time before you can do a DESCRIBE of an UPDATE statement.

```bash
setenv IFX_UPDDESC value
```
value  is any non-NULL value.

A NULL value (here meaning that IFX_UPDDESC is not set) disables the describe-for-update feature. Any non-NULL value enables the feature.

**IFX_XASTDCOMPLIANCE_XAEND**

In earlier releases of IBM Informix database servers, an internal rollback of a global transaction freed the transaction. In releases later than XPS 8.40 and IDS 9.40, however, the default behavior after an internal rollback is not to free the global transaction until an explicit rollback, as required by the X/Open XA standard. By setting the DISABLE_B162428_XA_FIX configuration parameter to 1, you can restore the legacy behavior as the default for all sessions.

The **IFX_XASTDCOMPLIANCE_XAEND** environment variable can override the configuration parameter for the current session, using the following syntax. Valid **IFX_XASTDCOMPLIANCE_XAEND** values are 1 and 0.

```
setenv IFX_XASTDCOMPLIANCE_XAEND 1 0
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Frees global transactions only after an explicit rollback</td>
</tr>
<tr>
<td>1</td>
<td>Frees global transactions after any rollback</td>
</tr>
</tbody>
</table>

This environment variable can be particularly useful when the server instance is disabled for new behavior by the DISABLE_B162428_XA_FIX configuration parameter, but one client requires the new behavior. Setting this environment variable to zero supports the new behavior in the current session.

**IFX_XFER_SHMBASE**

An alternate base address for a utility to attach the server shared memory segments.

```
setenv IFX_XFER_SHMBASE address
```

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid address in hexadecimal</td>
<td></td>
</tr>
</tbody>
</table>

After the database server allocates shared memory, the database server might allocate multiple contiguous OS shared memory segments. The client utility that connects to shared memory must attach all those OS segments contiguously also. The utility might have some other shared objects (for example, the xbsa library in onbar) loaded at the address where the server has shared memory segment attached. To workaround this situation, you can specify a different base address in the environment variable IFX_XFER_SHMBASE for the utility to attach the shared memory segments. The onstat, onmode, and oncheck utilities must attach to exact same shared memory base as oninit. Setting IFX_XFER_SHMBASE is not an option for these utilities.

**IMCADMIN**

The **IMCADMIN** environment variable supports the **imcadmin** administrative tool by specifying the name of a database server through which **imcadmin** can connect to MaxConnect. For **imcadmin** to operate correctly, you must set IMCADMIN before you use an IBM Informix product.
**IMCCONFIG**

The IMCCONFIG environment variable specifies a nondefault filename, and optionally a pathname, for the MaxConnect configuration file. On UNIX systems that support the C shell, this variable can be set by the following command.

```bash
setenv IMCCONFIG pathname
```

*pathname* is a full pathname or a simple filename.

When the setting is a filename that is not qualified by a full pathname, MaxConnect searches for the specified file in the `$INFORMIXDIR/etc/` directory. Thus, if you set IMCCONFIG to `IMCconfig.imc2`, MaxConnect searches for `$INFORMIXDIR/etc/IMCconfig.imc2` as its configuration file.

If the IMCCONFIG environment variable is not set, MaxConnect searches by default for `$INFORMIXDIR/etc/IMCconfig` as its configuration file.

**IMCSERVER**

The IMCSERVER environment variable specifies the name of a database server entry in the sqlhosts file that contains information on connectivity.

The database server can be either local or remote. On UNIX systems that support the C shell, the IMCSERVER environment variable can be set by the command.

```bash
setenv IMCSERVER dbservername
```

*dbservername* is the valid name of a database server.

Here *dbservername* must be the name of a database server in the sqlhosts file. For more information about sqlhosts settings with MaxConnect, see your IBM Informix Administrator's Guide. You cannot use MaxConnect unless IMCSERVER is set to a valid database server name.
INFORMIXC (UNIX)

The INFORMIXC environment variable specifies the filename or pathname of the C compiler to be used to compile files that IBM Informix ESQL/C generates. The setting takes effect only during the C compilation stage.

If INFORMIXC is not set, the default compiler on most systems is cc.

Tip: On Windows, you pass either -mcc or -bcc options to the esql preprocessor to use either the Microsoft or Borland C compilers.

```
$ setenv INFORMIXC compiler pathname
```

*compiler* is the filename of the C compiler.

*pathname* is the full pathname of the C compiler.

For example, to specify the GNU C compiler, enter the following command:

```
$ setenv INFORMIXC gcc
```

Important: If you use gcc, be aware that the database server assumes that strings are writable, so you need to compile using the -fwrable-strings option. Failure to do so can produce unpredictable results, possibly including core dumps.

INFORMIXCONCSMCFG (IDS)

The INFORMIXCONCSMCFG environment variable specifies the location of the concsm.cfg file that describes communications support modules.

```
$ setenv INFORMIXCONCSMCFG pathname
```

*pathname* specifies the full pathname of the concsm.cfg file.

The following command specifies that the concsm.cfg file is in */usr/myfiles*:

```
$ setenv INFORMIXCONCSMCFG /usr/myfiles
```

You can also specify a different name for the file. The following example specifies a filename of csmconfig in the same directory:

```
$ setenv INFORMIXCONCSMCFG /usr/myfiles/csmconfig
```

The default location of the concsm.cfg file is in $INFORMIXDIR/etc. For more information about communications support modules and the contents of the concsm.cfg file, refer to the IBM Informix Administrator’s Reference.

INFORMIXCONRETRY

The INFORMIXCONRETRY environment variable sets the maximum number of additional connection attempts that should be made to each database server by the client during the time limit that INFORMIXCONTIME specifies.

```
$ setenv INFORMIXCONRETRY count
```

*count* is the number of additional attempts to connect to each database server.
For example, the following command sets `INFORMIXCONRETRY` to specify three additional connection attempts (after the initial attempt):

```
setenv INFORMIXCONRETRY 3
```

The default value for `INFORMIXCONRETRY` is one retry after the initial connection attempt. The `INFORMIXCONTIME` setting, described in the following section, takes precedence over the `INFORMIXCONRETRY` setting.

**INFORMIXCONTIME**

The `INFORMIXCONTIME` environment variable specifies for how many seconds the CONNECT statement continues each attempt to establish a connection to a database server before returning an error. If you set no value, the default of 60 seconds can typically support a few hundred concurrent client connections, but some systems might encounter very few connection errors with a value as low as 15. The total distance between nodes, hardware speed, the volume of traffic, and the concurrency level of the network can all affect what value you should set to optimize `INFORMIXCONTIME`.

The `INFORMIXCONTIME` and `INFORMIXCONRETRY` environment variables let you configure your client-side connection capability to retry the connection instead of returning a `-908` error.

```
>>> setenv INFORMIXCONTIME seconds
```

`seconds` represents the minimum number of seconds spent in attempts to establish a connection to a database server.

For example, enter this command to set `INFORMIXCONTIME` to 60 seconds:

```
setenv INFORMIXCONTIME 60
```

If `INFORMIXCONTIME` is set to 60 and `INFORMIXCONRETRY` is set to 3, attempts to connect to the database server (after the initial attempt at 0 seconds) are made at 20, 40, and 60 seconds, if necessary, before aborting. This 20-second interval is the result of `INFORMIXCONTIME` divided by `INFORMIXCONRETRY`. If you attempt to set `INFORMIXCONTIME` to zero, the database server automatically resets it to the default value of 60 seconds.

If execution of the CONNECT statement involves searching `DBPATH`, the following rules apply:

- All appropriate servers in the `DBPATH` setting are accessed at least once, even though the `INFORMIXCONTIME` value might be exceeded. Thus, the CONNECT statement might take longer than the `INFORMIXCONTIME` time limit to return an error that indicates connection failure or that the database was not found.
- `INFORMIXCONRETRY` specifies how many additional connection attempts should be made for each database server entry in `DBPATH`.
- The `INFORMIXCONTIME` value is divided among the number of database server entries specified in `DBPATH`. Thus, if `DBPATH` contains numerous servers, you should increase the `INFORMIXCONTIME` value accordingly. For example, if `DBPATH` contains three entries, to spend at least 30 seconds attempting each connection, set `INFORMIXCONTIME` to 90.
INFORMIXCONTIME takes precedence over the INFORMIXCONRETRY setting. Retry efforts could end after the INFORMIXCONTIME value is exceeded, but before the INFORMIXCONRETRY value is reached.

The INFORMIXCONTIME and INFORMIXCONRETRY environment variables can be modified with the onutil SET command, as in the following example:

```
% onutil
1> SET INFORMIXCONTIME 120;
 Dynamic Configuration completed successfully
2> SET INFORMIXCONRETRY 10;
 Dynamic Configuration completed successfully
```

**INFORMIXCPPMAP (IDS)**

Set the INFORMIXCPPMAP environment variable to specify the fully qualified pathname of the map file for C++ programs. Information in the map file includes the database server type, the name of the shared library that supports the database object or value object type, the library entry point for the object, and the C++ library for which an object was built.

```
# setenv -g INFORMIXCPPMAP pathname
```

`pathname` is the directory path where the C++ map file is stored.

The map file is a text file that can have any filename. You can specify several map files, separated by colons (:) on UNIX or semicolons (;) on Windows.

On UNIX, the default map file is $INFORMIXDIR/etc/c++map. On Windows, the default map file is %INFORMIXDIR%\etc\c++map.

**INFORMIXDIR**

The INFORMIXDIR environment variable specifies the directory that contains the subdirectories in which your product files are installed. You must always set INFORMIXDIR. Verify that INFORMIXDIR is set to the full pathname of the directory in which you installed your database server. If you have multiple versions of a database server, set INFORMIXDIR to the appropriate directory name for the version that you want to access. For information about when to set INFORMIXDIR, see your IBM Informix Installation Guide.

```
# setenv -g INFORMIXDIR pathname
```

`pathname` is the directory path where the product files are installed.

To set INFORMIXDIR to /usr/informix/, for example, as the installation directory, enter the following command:

```
setenv INFORMIXDIR /usr/informix
```

**INFORMIXKEYTAB (UNIX)**

The INFORMIXKEYTAB environment variable specifies the location of the keytab file. The keytab file contains authentication information that database servers and clients access at connection time, if they use the DCE-GSS communications support module (CSM). It contains key tables that store keys, each of which contains a principal name (database server or user name), type, version, and value.
The database server uses the **keytab** file to find the key to register the database server and to acquire a credential for it. A client application uses the key if the user did not execute **dce_login** with the current operating-system user name (which is the same as the DCE principal name) or did not explicitly provide a credential.

```bash
setenv INFORMIXKEYTAB pathname
```

*pathname* specifies the full path of the **keytab** file.

For example, the following command specifies that the name and location of the **keytab** file is `/usr/myfiles/mykeytab`:

```bash
setenv INFORMIXKEYTAB /usr/myfiles/mykeytab
```

For more information about the DCE-GSS communications support module, see the *IBM Informix Administrator's Guide*.

**INFORMIXOPCACHE (IDS)**

The **INFORMIXOPCACHE** environment variable can specify the size of the memory cache for the staging-area blobspace of the client application.

```bash
setenv INFORMIXOPCACHE kilobytes
```

*kilobytes* specifies the value you set for the optical memory cache.

Set the **INFORMIXOPCACHE** environment variable by specifying the size of the memory cache in kilobytes. The specified size must be equal to or smaller than the size of the system-wide configuration parameter, **OPCACHEMAX**.

If you do not set **INFORMIXOPCACHE**, the default cache size is 128 kilobytes or the size specified in the configuration parameter **OPCACHEMAX**. The default for **OPCACHEMAX** is 128 kilobytes. If you set **INFORMIXOPCACHE** to a value of 0, Optical Subsystem does not use the cache.

**INFORMIXSERVER**

The **INFORMIXSERVER** environment variable specifies the default database server to which an explicit or implicit connection is made by an SQL API client, the DB-Access utility, or other IBM Informix products. This must be set before you can use IBM Informix client products. It has the following syntax.

```bash
setenv INFORMIXSERVER dbname
```

*dbname* is the name of the default database server.

The value of **INFORMIXSERVER** can be a local or remote server, but must correspond to a valid *dbname* entry in the `INFORMIXDIR/etc/sqlhosts` file on the computer running the application. The *dbname* must begin with a lower-case letter and cannot exceed 128 bytes. It can include any printable characters except uppercase characters, field delimiters (blank space or tab), the newline character, and the hyphen (or minus) symbol.

For example, this command specifies the **coral** database server as the default:

```bash
setenv INFORMIXSERVER coral
```
**INFORMIXSERVER** specifies the database server to which an application connects if the CONNECT DEFAULT statement is executed. It also defines the database server to which an initial implicit connection is established if the first statement in an application is not a CONNECT statement.

**Important:** You must set **INFORMIXSERVER** even if the application or DB-Access does not use implicit or explicit default connections.

For Extended Parallel Server, the **INFORMIXSERVER** environment variable specifies the name of a dbserver group. To specify a coserver name, use the following format:

```
dbservername.coserver_number
```

Here `dbservername` is the value that you assigned to the DBSERVERNAME configuration parameter in the ONCONFIG configuration file and `coserver_number` is the value that you assigned to the COSERVER configuration parameter for the connection coserver.

Strictly speaking, **INFORMIXSERVER** is not required for initialization. If **INFORMIXSERVER** is not set, however, Extended Parallel Server does not build the `sysmaster` tables.

**INFORMIXSHMBASE (UNIX)**

The **INFORMIXSHMBASE** environment variable affects only client applications connected to Informix databases that use the interprocess communications (IPC) shared-memory (ipcshtm) protocol.

**Important:** Resetting **INFORMIXSHMBASE** requires a thorough understanding of how the application uses memory. Normally you do not reset **INFORMIXSHMBASE**.

**INFORMIXSHMBASE** specifies where shared-memory communication segments are attached to the client process so that client applications can avoid collisions with other memory segments that it uses. If you do not set **INFORMIXSHMBASE**, the memory address of the communication segments defaults to an implementation-specific value such as `0x800000`.

```
setenv INFORMIXSHMBASE value
```

`value` is an integer (in kilobytes) used to calculate the memory address.

The database server calculates the memory address where segments are attached by multiplying the value of **INFORMIXSHMBASE** by 1,024. For example, on a system that uses the C shell, you can set the memory address to the value `0x800000` by entering the following command:

```
setenv INFORMIXSHMBASE 8192
```

For more information, see your IBM Informix Administrator’s Guide and the IBM Informix Administrator’s Reference.

**INFORMIXSQLHOSTS**

The **INFORMIXSQLHOSTS** environment variable specifies where the SQL client or the database server can find connectivity information.
On UNIX systems, the default search path for the connectivity information file is
$INFORMIXDIR/etc/sqlhosts.

The following command overrides this default to specify the mysqlhosts file in the
/work/envt directory:
setenv INFORMIXSQLHOSTS /work/envt/mysqlhosts

On Windows, INFORMIXSQLHOSTS points to the computer whose registry
contains the SQLHOSTS subkey.

The next example specifies that the client or database server look for connectivity
information on a computer named arizona:
set INFORMIXSQLHOSTS = \arizona

For details of the information that sqlhosts (or a file with a non-default filename)
can provide about connectivity, see your IBM Informix Administrator’s Guide.

INFORMIXSTACKSIZE

The INFORMIXSTACKSIZE environment variable specifies the stack size (in
kilobytes) that is applied to all client processes. Any value that you set for
INFORMIXSTACKSIZE in the client environment is ignored by the database server.

setenv -STACKSIZE size

size is an integer, setting the stack size (in kilobytes) for SQL client
threads.

For example, to decrease the INFORMIXSTACKSIZE to 20 kilobytes, enter the
following command:
setenv -STACKSIZE 20

If INFORMIXSTACKSIZE is not set, the stack size is taken from the database
server configuration parameter STACKSIZE or else defaults to a platform-specific
value. The default stack size value for the primary thread of an SQL client is 32
kilobytes for nonrecursive database activity.

Warning: For instructions on setting this value, see the IBM Informix
Administrator’s Reference. If you incorrectly set the value of
INFORMIXSTACKSIZE, it can cause the database server to fail.

INFORMIXTERM (UNIX)

The INFORMIXTERM environment variable specifies whether DB–Access should
use the information in the termcap file or the terminfo directory.

On character-based systems, the termcap file and terminfo directory determine
terminal-dependent keyboard and screen capabilities, such as the operation of
function keys, color and intensity attributes in screen displays, and the definition
of window borders and graphic characters.

pathname is the full pathname of the connectivity information file.
If `INFORMIXTERM` is not set, the default setting is `termcap`. When DB–Access is installed on your system, a `termcap` file is placed in the `etc` subdirectory of `$INFORMIXDIR`. This file is a superset of an operating-system `termcap` file.

You can use the `termcap` file that the database server supplies, the system `termcap` file, or a `termcap` file that you create. You must set the `TERMCP` environment variable if you do not use the default `termcap` file. For information on setting the `TERMCP` environment variable, see page 3-66.

The `terminfo` directory contains a file for each terminal name that has been defined. The `terminfo` setting for `INFORMIXTERM` is supported only on computers that provide full support for the UNIX System V `terminfo` library. For details, see the machine notes file for your product.

**INF_ROLE_SEP (IDS)**

The `INF_ROLE_SEP` environment variable configures the security feature of role separation when the database server is installed or reinstalled on UNIX systems. Role separation enforces separating administrative tasks by people who run and audit the database server. After the installation is complete, `INF_ROLE_SEP` has no effect. If `INF_ROLE_SEP` is not set, then user `informix` (the default) can perform all administrative tasks.

```
setenv INF_ROLE_SEP n
```

`n` is any positive integer.

On Windows, the install process asks whether you want to enable role separation regardless of the setting of `INF_ROLE_SEP`. To enable role separation for database servers on Windows, choose the role-separation option during installation.

If `INF_ROLE_SEP` is set when Dynamic Server is installed on a UNIX platform, role separation is implemented and a separate group is specified to serve each of the following responsibilities:

- The Database Server Administrator (DBSA)
- The Audit Analysis Officer (AAO)
- The standard user

On UNIX, you can establish role separation publicationly by changing the group that owns the `aodir`, `dbsadir`, or `etc` directories at any time after the installation is complete. You can disable role separation by resetting the group that owns these directories to `informix`. You can have role separation enabled, for example, for the Audit Analysis Officer (AAO) without having role separation enabled for the Database Server Administrator (DBSA).

For more information about the security feature of role separation, see the *IBM Informix Security Guide*. To learn how to configure role separation when you install your database server, see your *IBM Informix Installation Guide*. 
**INTERACTIVE_DESKTOP_OFF (Windows)**

This environment variable lets you prevent interaction with the Windows desktop when an SPL routine executes a SYSTEM command.

```
>>> setenv INTERACTIVE_DESKTOP_OFF 1
```

If **INTERACTIVE_DESKTOP_OFF** is 1 and an SPL routine attempts to interact with the desktop (for example, with the `notepad.exe` or `cmd.exe` program), the routine fails unless the user is a member of the **Administrators** group.

The valid settings (1 or 0) have the following effects:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prevents the database server from acquiring desktop resources for the user executing the stored procedure</td>
</tr>
<tr>
<td>0</td>
<td>SYSTEM commands in a stored procedure can interact with the desktop. This is the default value.</td>
</tr>
</tbody>
</table>

Setting **INTERACTIVE_DESKTOP_OFF** to 1 allows an SPL routine that does not interact with the desktop to execute more quickly. This setting also allows the database server to simultaneously call a greater number of SYSTEM commands because the command no longer depends on a limited operating-system resource (Desktop and WindowStation handles).

**ISM_COMPRESSION**

Set this environment variable in the ON-Bar environment to specify whether the IBM Informix Storage Manager (ISM) should use data compression.

```
>>> setenv ISM_COMPRESSION TRUE
```

If **ISM_COMPRESSION** is set to **TRUE** in the environment of the ON-Bar process that makes a request, the ISM server uses a data-compression algorithm to store or retrieve the requested data. If **ISM_COMPRESSION** is set to **FALSE** or is not set, the ISM server does not use compression.

**ISM_DEBUG_FILE**

Set the **ISM_DEBUG_FILE** environment variable in the IBM Informix Storage Manager server environment to specify where to write XBSA messages.

```
>>> setenv ISM_DEBUG_FILE pathname
```

*pathname* specifies the location of the XBSA message log file.

If you do not set **ISM_DEBUG_FILE**, the XBSA message log is located in the `$INFORMIXDIR/ism/applogs/xbsa.messages` directory on UNIX, or in the `c:\nsr\applogs\xbsa.messages` directory on Windows systems.
**ISM_DEBUG_LEVEL**

Set the **ISM_DEBUG_LEVEL** environment variable in the ON–Bar environment to control the level of reporting detail recorded in the XBSA messages log. The XBSA shared library writes to this log.

```
>>> setenv ISM_DEBUG_LEVEL value
```

`value` specifies the level of reporting detail, where $1 \leq value \leq 9$.

If **ISM_DEBUG_LEVEL** is not set, has a null value, or has a value outside this range, the default detail level is 1. A detail level of 0 suppresses all XBSA debugging records. A detail level of 1 reports only XBSA failures.

**ISM_ENCRYPTION**

Set the **ISM_ENCRYPTION** environment variable in the ON–Bar environment to specify whether IBM Informix Storage Manager (ISM) uses data encryption.

```
>>> setenv ISM_ENCRYPTION XOR
```

Three settings of **ISM_ENCRYPTION** are supported:

- XOR uses encryption.
- NONE does not use encryption.
- TRUE uses encryption.

If **ISM_ENCRYPTION** is set to NONE or is not set, the ISM server does not use encryption.

If the **ISM_ENCRYPTION** is set to TRUE or XOR in the environment of the ON–Bar process that makes a request, the ISM server uses encryption to store or retrieve the data specified in that request.

**ISM_MAXLOGSIZE**

Set the **ISM_MAXLOGSIZE** environment variable in the IBM Informix Storage Manager (ISM) server environment to specify the size threshold of the ISM activity log.

```
>>> setenv ISM_MAXLOGSIZE size
```

`size` specifies the size threshold (in megabytes) of the activity log.

If **ISM_MAXLOGSIZE** is not set, then the default size limit is 1 megabyte. If **ISM_MAXLOGSIZE** is set to a null value, then the threshold is 0 bytes.

**ISM_MAXLOGVERS**

Set the **ISM_MAXLOGVERS** environment variable in the IBM Informix Storage Manager (ISM) server environment to specify the maximum number of activity-log files to be preserved by the ISM server.
value specifies the number of files to be preserved.

If ISM_MAXLOGVERS is not set, then the default number of files is four. If the setting is a null value, then the ISM server preserves no activity log files.

### JAR_TEMP_PATH (IDS)
Set the JAR_TEMP_PATH variable to specify a non-default local file system location where jar management procedures such as install_jar() and replace_jar() can store temporary .jar files of the Java virtual machine.

pathname specifies a local directory for temporary .jar files.

This directory must have read and write permissions for the user who brings up the database server. If the JAR_TEMP_PATH environment variable is not set, temporary copies of .jar files are stored in the /tmp directory of the local file system for the database server.

### JAVA_COMPILER (IDS)
You can set the JAVA_COMPILER environment variable in the Java virtual machine environment to disable JIT compilation.

The NONE and none settings are equivalent. On UNIX systems that support the C shell and on which JAVA_COMPILER has been set to NONE or none, you can enable the JIT compiler for the JVM environment by the following command:

```bash
unset JAVA_COMPILER
```

### JVM_MAX_HEAP_SIZE (IDS)
The JVM_MAX_HEAP_SIZE environment variable can set a non-default upper limit on the size of the heap for the Java virtual machine.

size is a positive integer that specifies the maximum size (in megabytes).

For example, the following command sets the maximum heap size at 12 MB:

```bash
set JVM_MAX_HEAP_SIZE 12
```

If you do not set JVM_MAX_HEAP_SIZE, 16 MB is the default maximum size.
**LD_LIBRARY_PATH (UNIX)**

The **LD_LIBRARY_PATH** environment variable tells the shell on Solaris systems which directories to search for client or shared Informix general libraries. You must specify the directory that contains your client libraries before you can use the product.

```
setenv LD_LIBRARY_PATH $PATH:pathname
```

*pathname* specifies the search path for the library.

For INtersolv DataDirect ODBC Driver on AIX®, set **LIBPATH**. For INtersolv DataDirect ODBC Driver on HP-UX, set **SHLIB_PATH**.

The following example sets the **LD_LIBRARY_PATH** environment variable to the desired directory:

```
setenv LD_LIBRARY_PATH
$INFORMIXDIR/lib:$INFORMIXDIR/lib/esql:$LD_LIBRARY_PATH
```

**LIBERAL_MATCH (XPS)**

The **LIBERAL_MATCH** environment variable allows the database server to ignore trailing blanks when the LIKE and MATCHES operators occur in SQL statements that compare two column values.

```
setenv LIBERAL_MATCH
```

When this environment variable is set, the database server ignores trailing blanks in a LIKE or MATCHES condition. For example, if **LIBERAL_MATCH** is set, and you specify "M LIKE P" when P contains trailing blank spaces that do not occur in M, the result is TRUE. When this environment variable is not set, the database server returns FALSE for string comparisons like this that differ only in trailing blank characters.

This environment variable supports behavior consistent with that of the LIKE and MATCHES operators in Dynamic Server, Versions 7.x, 9.x, and 10.x. This behavior (like the MATCHES operator) is an extension to the ANSI/ISO standard for SQL.

For more information about the LIKE and MATCHES operators, refer to the *IBM Informix Guide to SQL: Syntax*.

**LIBPATH (UNIX)**

The **LIBPATH** environment variable tells the shell on AIX systems which directories to search for dynamic-link libraries for the INtersolv DataDirect ODBC Driver. You must specify the full pathname for the directory where you installed the product.

```
setenv LIBPATH pathname
```

*pathname* specifies the search path for the libraries.
On Solaris, set `LD_LIBRARY_PATH`. On HP-UX, set `SHLIB_PATH`.

**NODEFDAC**

When the `NODEFDAC` environment variable is set to `yes`, it prevents default table privileges (Select, Insert, Update, and Delete) from being granted to PUBLIC when a new table is created during the current session in a database that is not ANSI compliant.

```
$ setenv NODEFDAC yes
```

The `yes` setting is case sensitive, and is also sensitive to leading and trailing blank spaces. Including uppercase letters or blank spaces in the setting is equivalent to leaving `NODEFDAC` unset. When `NODEFDAC` is not set, or if it is set to any value besides `yes`, default privileges on tables and Owner-mode UDRs are granted to PUBLIC by default when the table or UDR is created in a database that is not ANSI-compliant.

**ONCONFIG**

The `ONCONFIG` environment variable specifies the name of the active file that holds configuration parameters for the database server. This file is read as input during the initialization procedure. After you prepare the `ONCONFIG` configuration file, set `ONCONFIG` to the name of this file.

```
$ setenv ONCONFIG $filename
```

`filename` is the name of a file in `$INFORMIXDIR/etc` that contains the configuration parameters for your database.

To prepare the `ONCONFIG` file, make a copy of the `onconfig.std` file and modify the copy. It is recommended that you name the `ONCONFIG` file so that it can easily be related to a specific database server. If you have multiple instances of a database server, each instance must have its own uniquely named `ONCONFIG` file.

To prepare the `ONCONFIG` file for Extended Parallel Server, make a copy of the `onconfig.std` file if you are using a single coserver configuration or make a copy of the `onconfig.xps` file if you are using a multiple coserver configuration. You can use the `onconfig.std` file for a multiple coserver configuration, but you would have to add additional keywords and configuration parameters such as END, NODE, and COSERVER, which are already provided for you in the `onconfig.xps` file.

If the `ONCONFIG` environment variable is not set, the database server uses configuration values from either the `SONCONFIG` file or the `$INFORMIXDIR/etc/onconfig` file.

For more information on configuration parameters and the `ONCONFIG` file, see the *IBM Informix Administrator’s Reference*. 
**OPTCOMPIND**

You can set the **OPTCOMPIND** environment variable so that the optimizer can select the appropriate join method.

```
setenv OPTCOMPIND 2
```

0  A nested-loop join is preferred, where possible, over a sort-merge join or a hash join.
1  When the isolation level is not Repeatable Read, the optimizer behaves as in setting 2; otherwise, the optimizer behaves as in setting 0.
2  Nested-loop joins are not necessarily preferred. The optimizer bases its decision purely on costs, regardless of transaction isolation mode.

When **OPTCOMPIND** is not set, the database server uses the OPTCOMPIND value from the ONCONFIG configuration file. When neither the environment variable nor the configuration parameter is set, the default value is 2.

On Dynamic Server, the SET ENVIRONMENT OPTCOMPIND statement can set or reset **OPTCOMPIND** dynamically at runtime. This overrides the current **OPTCOMPIND** value (or the ONCONFIG configuration parameter OPTCOMPIND) for the current user session only. For more information on the SET ENVIRONMENT OPTCOMPIND statement of SQL see the *IBM Informix Guide to SQL: Syntax*.

For more information on the ONCONFIG configuration parameter OPTCOMPIND, see the *IBM Informix Administrator’s Reference*. For more information on the different join methods that the optimizer uses, see your *IBM Informix Performance Guide*.

**OPTMSG**

Set the **OPTMSG** environment variable at runtime before you start an IBM Informix ESQL/C application to enable (or disable) optimized message transfers (message chaining) for all SQL statements in an application.

```
setenv OPTMSG 1
```

0  disables optimized message transfers.
1  enables optimized message transfers and implements the feature for any subsequent connection.

The default value is 0 (zero), which explicitly disables message chaining. You might want, for example, to disable optimized message transfers for statements that require immediate replies, for debugging, or to ensure that the database server processes all messages before the application terminates.
When you set `OPTMSG` within an application, you can activate or deactivate optimized message transfers for each connection or within each thread. To enable optimized message transfers, you must set `OPTMSG` before you establish a connection.

For more information about setting `OPTMSG` and defining related global variables, see the *IBM Informix ESQL/C Programmer’s Manual*.

**OPTOFC**

Set the `OPTOFC` environment variable to enable optimize-OPEN-FETCH-CLOSE functionality in an IBM Informix ESQL/C application or other APIs (such as JDBC, ODBC, OLE DB, LIBDMI, and Lib C++) that use DECLARE and OPEN statements to execute a cursor.

```bash
->setenv OPTOFC 0
```

- 0 disables `OPTOFC` for all threads of the application.
- 1 enables `OPTOFC` for every cursor in every thread of the application.

The default value is 0 (zero).

The `OPTOFC` environment variable reduces the number of message requests between the application and the database server.

If you set `OPTOFC` from the shell, you must set it before you start the ESQL/C application. For more information about enabling `OPTOFC` and related features, see the *IBM Informix ESQL/C Programmer’s Manual*.

**OPT_GOAL (IDS, UNIX)**

Set the `OPT_GOAL` environment variable in the user environment, before you start an application, to specify the query performance goal for the optimizer.

```bash
->setenv OPT_GOAL 1
```

- 0 specifies user-response-time optimization.
- -1 specifies total-query-time optimization.

The default behavior is for the optimizer to choose query plans that optimize the total query time.

You can also specify the optimization goal for individual queries with optimizer directives or for a session with the `SET OPTIMIZATION` statement.

Both methods take precedence over the `OPT_GOAL` environment variable setting. You can also set the `OPT_GOAL` configuration parameter for the Dynamic Server system; this method has the lowest level of precedence.

For more information about optimizing queries for your database server, see your *IBM Informix Performance Guide*. For information on the `SET OPTIMIZATION` statement, see the *IBM Informix Guide to SQL: Syntax*. 
PATH

The UNIX PATH environment variable tells the shell which directories to search for executable programs. You must add the directory containing your IBM Informix product to your PATH setting before you can use the product.

```
setenv PATH $PATH:pathname
```

*pathname* specifies the search path for the executables.

Include a colon (:) separator between the pathnames on UNIX systems. (Use the semicolon (;) separator between pathnames on Windows systems.)

You can specify the search path in various ways. The PATH environment variable tells the operating system where to search for executable programs. You must include the directory that contains your IBM Informix product in your path setting before you can use the product. This directory should appear before $INFORMIXDIR/bin, which you must also include.

For additional information about how to modify your path, see "Modifying an Environment-Variable Setting" on page 3-6.

PDQPRIORITY

For Dynamic Server, the PDQPRIORITY environment variable determines the degree of parallelism that the database server uses and affects how the database server allocates resources, including memory, processors, and disk reads.

For Extended Parallel Server, the PDQPRIORITY environment variable determines only the allocation of memory resources.

```
setenv PDQPRIORITY HIGH
```

(1)

*resources* Is an integer in the range 0 to 100. Value 1 is the same as LOW.

Value 0 is the same as OFF (for Dynamic Server only).

*high_value* Optional integer value that requests the maximum percentage of memory (for Extended Parallel Server only). When you specify this value after the resources value, you request a range of memory, expressed as a percentage.

Here the HIGH, LOW, and OFF keywords have the following effects:

**HIGH** When the database server allocates resources among all users, it gives as many resources as possible to the query.

**LOW** Data values are fetched from fragmented tables in parallel.

Notes:

1. Extended Parallel Server only
OFF PDQ processing is turned off (for Dynamic Server only).

Usually, the more resources a database server uses, the better its performance for a given query. If the server uses too many resources, however, contention for the resources can take resources away from other queries, resulting in degraded performance. For more information on performance considerations for PDQPRIORITY, refer to your IBM Informix Performance Guide.

An application can override the setting of this environment variable when it issues the SQL statement SET PDQPRIORITY, as the IBM Informix Guide to SQL: Syntax describes.

**Using PDQPRIORITY with Dynamic Server**

The resources value specifies the query priority level and the amount of resources that the database server uses to process the query. When you specify LOW, Dynamic Server uses no forms of parallelism.

When PDQPRIORITY is not set, the default value is OFF.

When PDQPRIORITY is set to HIGH, Dynamic Server determines an appropriate value to use for PDQPRIORITY based on several criteria. These include the number of available processors, the fragmentation of tables queried, the complexity of the query, and additional factors.

**Using PDQPRIORITY with Extended Parallel Server**

The resources value establishes the minimum percentage of memory when you also specify high_value to request a range of memory allocation. Other parallel operations can occur when the PDQPRIORITY setting is LOW.

When the PDQPRIORITY environment variable is not set, the default value is the value of the PDQPRIORITY configuration parameter.

When PDQPRIORITY is set to 0, Extended Parallel Server can execute a query in parallel, depending on the number of available processors, the fragmentation of tables queried, the complexity of the query, and other factors. PDQPRIORITY does not affect the degree of parallelism in Extended Parallel Server.

An application can prevent changes to the PDQPRIORITY setting with the SET PDQPRIORITY IMMUTABLE or SET ALL_MUTABLES statements of SQL. You can also override the setting of this environment variable by issuing the SQL statement SET ENVIRONMENT to change the IMPLICIT_PDQ or BOUNT_IMPL_PDQ options, as the IBM Informix Guide to SQL: Syntax describes.

**PLCONFIG (IDS)**

The PLCONFIG environment variable specifies the name of the configuration file that the High-Performance Loader (HPL) uses. This file must reside in the $INFORMIXDIR/etc directory. If the PLCONFIG environment variable is not set, then $INFORMIXDIR/etc/plconfig is the default configuration file.

```bash
setenv PLCONFIG filename
```

**filename** specifies the simple filename of the configuration file that the High-Performance Loader uses.
For example, to specify the `$INFORMIXDIR/etc/custom.cfg` file as the configuration file for the High-Performance Loader, enter the following command:

```
setenv PLCONFIG custom.cfg
```

For more information, see the **IBM Informix High-Performance Loader User’s Guide**.

**PLOAD_LO_PATH (IDS)**

The `PLOAD_LO_PATH` environment variable lets you specify the pathname for smart-large-object handles (which identify the location of smart large objects such as BLOB and CLOB data types).

```
setenv PLOAD_LO_PATH pathname
```

`pathname` specifies the directory for the smart-large-object handles.

If `PLOAD_LO_PATH` is not set, the default directory is `/tmp`.

For more information, see the **IBM Informix High-Performance Loader User’s Guide**.

**PLOAD_SHMBASE (IDS)**

The `PLOAD_SHMBASE` environment variable lets you specify the shared-memory address at which the High-Performance Loader (HPL) `onpload` processes will attach. If `PLOAD_SHMBASE` is not set, the HPL determines which shared-memory address to use.

```
setenv PLOAD_SHMBASE value
```

`value` is used to calculate the shared-memory address.

If the `onpload` utility cannot attach, an error appears and you must specify a new value.

The `onpload` utility tries to determine at which address to attach, as follows in the following (descending) order:

1. Attach at the same address (SHMBASE) as the database server.
2. Attach beyond the database server segments.
3. Attach at the address specified in `PLOAD_SHMBASE`.

Tip: It is recommended that you let the HPL decide where to attach and that you set `PLOAD_SHMBASE` only if necessary to avoid shared-memory collisions between `onpload` and the database server.

For more information, see the **IBM Informix High-Performance Loader User’s Guide**.

**PSORT_DBTEMP**

The `PSORT_DBTEMP` environment variable specifies where the database server writes the temporary files it uses when it performs a sort.

```
setenv PSORT_DBTEMP pathname
```
pathname is the name of the UNIX directory used for intermediate writes during a sort.

To set the PSORT_DBTEMP environment variable to specify the directory (for example, /usr/leif/tempsort), enter the following command:

```
setenv PSORT_DBTEMP /usr/leif/tempsort
```

For maximum performance, specify directories that reside in file systems on different disks.

You might also want to consider setting the environment variable DBSPACETEMP to place temporary files used in sorting in dbspaces rather than operating-system files. See the discussion of the DBSPACETEMP environment variable in "DBSPACETEMP" on page 3-29.

The database server uses the directory that PSORT_DBTEMP specifies, even if the environment variable PSORT_NPROCS is not set. For additional information about the PSORT_DBTEMP environment variable, see your IBM Informix Administrator's Guide and your IBM Informix Performance Guide.

**PSORT_NPROCS**

The PSORT_NPROCS environment variable enables the database server to improve the performance of the parallel-process sorting package by allocating more threads for sorting.

PSORT_NPROCS does not necessarily improve sorting speed for Extended Parallel Server, because the database server sorts in parallel whether this environment variable is set or not.

Before the sorting package performs a parallel sort, make sure that the database server has enough memory for the sort.

```
setenv PSORT_NPROCS threads
```

threads is an integer, specifying the maximum number of threads to be used to sort a query. This value cannot be greater than 10.

The following command sets PSORT_NPROCS to 4:

```
setenv PSORT_NPROCS 4
```

To disable parallel sorting, enter the following command:

```
unsetenv PSORT_NPROCS
```

It is recommended that you initially set PSORT_NPROCS to 2 when your computer has multiple CPUs. If subsequent CPU activity is lower than I/O activity, you can increase the value of PSORT_NPROCS.

**Tip:** If the PDQPRIORITY environment variable is not set, the database server allocates the minimum amount of memory to sorting. This minimum memory is insufficient to start even two sort threads. If you have not set PDQPRIORITY, check the available memory before you perform a large-scale sort (such as an index build) to make sure that you have enough memory.
**Default Values for Detached Indexes**
If the `PSORT_NPROCS` environment variable is set, the database server uses the specified number of sort threads as an upper limit for ordinary sorts. If `PSORT_NPROCS` is not set, parallel sorting does not take place. The database server uses one thread for the sort. If `PSORT_NPROCS` is set to 0, the database server uses three threads for the sort.

**Default Values for Attached Indexes**
The default number of threads is different for attached indexes.

If the `PSORT_NPROCS` environment variable is set, you get the specified number of sort threads for each fragment of the index that is being built.

If `PSORT_NPROCS` is not set, or if it is set to 0, you get two sort threads for each fragment of the index unless you have a single-CPU virtual processor. If you have a single-CPU virtual processor, you get one sort thread for each fragment of the index.

For additional information about the `PSORT_NPROCS` environment variable, see your `IBM Informix Administrator’s Guide` and your `IBM Informix Performance Guide`.

**RTREE_COST_ADJUST_VALUE (IDS)**
The `RTREE_COST_ADJUST_VALUE` environment variable specifies a coefficient that support functions of user-defined data types can use to estimate the cost of an R-tree index for queries on UDT columns.

```
setenv RTREE_COST_ADJUST_VALUE value
```

`value` is a floating-point number, where `1 ≤ value ≤ 1000`, specifying a multiplier for estimating the cost of using an index on a UDT column.

For spatial queries, the I/O overhead tends to exceed by far the CPU cost, so by multiplying the uncorrected estimated cost by an appropriate `value` from this setting, the database server can make better cost-based decisions on how to implement queries on UDT columns for which an R-tree index exists.

**SHLIB_PATH (UNIX)**
The `SHLIB_PATH` environment variable tells the shell on HP-UX systems which directories to search for dynamic-link libraries. This is used, for example, with the INTERSOLV DataDirect ODBC Driver. You must specify the full pathname for the directory where you installed the product.

```
setenv SHLIB_PATH $PATH:pathname
```

`pathname` specifies the search path for the libraries.

On Solaris systems, set `LD_LIBRARY_PATH`. On AIX systems, set `LIBPATH`. 
**STMT_CACHE (IDS)**

Use the STMT_CACHE environment variable to control the use of the shared-statement cache on a session. This feature can reduce memory consumption and can speed query processing among different user sessions. Valid STMT_CACHE values are 1 and 0.

```
$ setenv STMT_CACHE 1
```

1 enables the SQL statement cache.

0 disables the SQL statement cache.

Set the STMT_CACHE environment variable for applications that do not use the SET STMT_CACHE statement to control the use of the SQL statement cache. By default, a statement cache of 512 kilobytes is enabled, but this feature can be disabled or set to a non-default size through the STMT_CACHE parameter of the onconfig.std file or by the SET STMT_CACHE statement.

This environment variable has no effect if the SQL statement cache is disabled through the configuration file setting. Values set by the SET STMT_CACHE statement in the application override the STMT_CACHE setting.

**TERM (UNIX)**

The TERM environment variable is used for terminal handling. It lets DB–Access (and other character-based applications) recognize and communicate with the terminal that you are using.

```
$ setenv TERM type
```

`type` specifies the terminal type.

The terminal type specified in the TERM setting must correspond to an entry in the termcap file or terminfo directory.

Before you can set the TERM environment variable, you must obtain the code for your terminal from the database administrator.

For example, to specify the vt100 terminal, set the TERM environment variable by entering the following command:

```
setenv TERM vt100
```

**TERMCAP (UNIX)**

The TERMCAP environment variable is used for terminal handling. It tells DB–Access (and other character-based applications) to communicate with the termcap file instead of the terminfo directory.

```
$ setenv TERMCAP pathname
```

`pathname` specifies the location of the termcap file.
The termcap file contains a list of various types of terminals and their characteristics. For example, to provide DB–Access terminal-handling information, which is specified in the /usr/informix/etc/termcap file, enter the following command:

```bash
setenv TERMCAP /usr/informix/etc/termcap
```

You can use set TERMCAP in any of the following ways. If several termcap files exist, they have the following (descending) order of precedence:

1. The termcap file that you create
2. The termcap file that the database server supplies (that is, $INFORMIXDIR/etc/termcap)
3. The operating-system termcap file (that is, /etc/termcap)

If you set the TERMCAP environment variable, be sure that the INFORMIXTERM environment variable is set to the default, termcap.

If you do not set the TERMCAP environment variable, the system file (that is, /etc/termcap) is used by default.

### TERMINFO (UNIX)

The TERMINFO environment variable is used for terminal handling.

The environment variable is supported only on platforms that provide full support for the terminfo libraries that System V and Solaris UNIX systems provide.

```bash
setenv TERMINFO /usr/lib/terminfo
```

TERMINFO tells DB–Access to communicate with the terminfo directory instead of the termcap file. The terminfo directory has subdirectories that contain files that pertain to terminals and their characteristics.

To set TERMINFO, enter the following command:

```bash
setenv TERMINFO /usr/lib/terminfo
```

If you set the TERMINFO environment variable, you must also set the INFORMIXTERM environment variable to terminfo.

### THREADLIB (UNIX)

Use the THREADLIB environment variable to compile multithreaded ESQL/C applications. A multithreaded ESQL/C application lets you establish as many connections to one or more databases as there are threads. These connections can remain active while the application program executes.

The THREADLIB environment variable indicates which thread package to use when you compile an application. Currently only the Distributed Computing Environment (DCE) is supported.

```bash
setenv THREADLIB DCE
```

The THREADLIB environment variable is checked when the -thread option is passed to the ESQL/C script when you compile a multithreaded ESQL/C application.
application. When you use the -thread option while compiling, the ESQL/C script generates an error if THREADLIB is not set, or if THREADLIB is set to an unsupported thread package.

TOBIGINT (XPS)

You can use the TOBIGINT environment variable to change the default INT8 label that the dbschema utility displays in its output for columns of the INT8 data type to the string BIGINT.

```
>> setenv TOBIGINT 1
```

Set TOBIGINT to 1 to enable, and unset TOBIGINT to disable this dbschema functionality. The name BIGINT is the identifier of a built-in 8-byte integer data type of DB2® database servers of IBM. See the Migration Guide for additional information about the TOBIGINT environment variable.

USETABLEAME (IDS)

The USETABLEAME environment variable can prevent users from using a synonym to specify the table in ALTER TABLE or DROP TABLE statements. Unlike most environment variables, USETABLEAME does not need to be set to a value. It takes effect if you set it to any value, or to no value.

```
>> setenv USETABLEAME
```

By default, ALTER TABLE or DROP TABLE statements accept a valid synonym for the name of the table to be altered or dropped. (In contrast, RENAME TABLE issues an error if you specify a synonym, as do the ALTER SEQUENCE, DROP SEQUENCE, and RENAME SEQUENCE statements, if you attempt to substitute a synonym for the sequence name in those statements.)

If you set USETABLEAME, an error results if a synonym appears in ALTER TABLE or DROP TABLE statements. Setting USETABLEAME has no effect on the DROP VIEW statement, which accepts a valid synonym for the view.

XFER_CONFIG (XPS)

The XFER_CONFIG environment variable specifies the location of the xfer_config configuration file.

```
>> setenv XFER_CONFIG pathname
```

pathname specifies the location of the xfer_config file.

The xfer_config file works with the onxfer utility to help users migrate from Version 7.x to Version 8.x. It contains various configuration parameter settings that users can modify and a list of tables that users can select to be transferred.

The default xfer_config file is located in the $INFORMIXDIR/etc directory on UNIX systems or in the %INFORMIXDIR%\etc directory in Windows.
Index of Environment Variables

Table 3-4 on page 3-69 provides an overview of the uses for the various Informix and UNIX environment variables. This serves as an index to general topics and lists the related environment variables and the pages where the environment variables are introduced. Where the Topic column is empty, the entry refers to the previously listed topic.

The term GLS Guide in the Page column in [Table 3-4] indicates environment variables that are described in the IBM Informix GLS User’s Guide.

The term ER Guide in the Page column in [Table 3-4] indicates environment variables that are described in the IBM Informix Dynamic Server Enterprise Replication Guide.

Table 3-4. Uses for Environment Variables

<table>
<thead>
<tr>
<th>Topic</th>
<th>Environment Variable</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abbreviated year values</td>
<td>DBCENTURY</td>
<td>3-18</td>
</tr>
<tr>
<td>Alarms for SQL operations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Globally detached indexes</td>
<td>GLOBAL_DETACH_INFRM</td>
<td>3-36</td>
</tr>
<tr>
<td>Cartesian joins</td>
<td>IFMX_CART_ALARM</td>
<td>3-38</td>
</tr>
<tr>
<td>ANSI/ISO SQL compliance</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lettercase of owner names</td>
<td>ANSIOWNER</td>
<td>3-14</td>
</tr>
<tr>
<td>Informix syntax extensions</td>
<td>DBANSIWARN</td>
<td>3-17</td>
</tr>
<tr>
<td>default table privileges</td>
<td>NODEFDAC</td>
<td>3-58</td>
</tr>
<tr>
<td>archchecker utility</td>
<td>AC_CONFIG</td>
<td>3-14</td>
</tr>
<tr>
<td>Buffer: fetch size</td>
<td>FET_BUF_SIZE</td>
<td>3-36</td>
</tr>
<tr>
<td>network size</td>
<td>IFX_NETBUF_SIZE</td>
<td>3-44</td>
</tr>
<tr>
<td>network pool size</td>
<td>IFX_NETBUF_PVTPOOL_SIZE</td>
<td>3-42</td>
</tr>
<tr>
<td>BYTE or TEXT data buffer</td>
<td>DBBLOBBUF</td>
<td>3-17</td>
</tr>
<tr>
<td>Cache: enabling</td>
<td>STMT_CACHE</td>
<td>3-66</td>
</tr>
<tr>
<td>size for Optical Subsystem</td>
<td>INFORMIXOPCACHE</td>
<td>3-50</td>
</tr>
<tr>
<td>Client/server:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>default server</td>
<td>INFORMIXSERVER</td>
<td>3-50</td>
</tr>
<tr>
<td>shared memory segments</td>
<td>INFORMIXSHMBASE</td>
<td>3-51</td>
</tr>
<tr>
<td>stacksize for client session</td>
<td>INFORMIXSTACKSIZE</td>
<td>3-52</td>
</tr>
<tr>
<td>locale of client, server</td>
<td>CLIENT_LOCALE, DB locale</td>
<td>GLS Guide</td>
</tr>
<tr>
<td>locale for file I/O</td>
<td>SERVER_LOCALE</td>
<td>GLS Guide</td>
</tr>
<tr>
<td>Code-set conversion</td>
<td></td>
<td></td>
</tr>
<tr>
<td>code set of client, server</td>
<td>CLIENT_LOCALE, DB locale</td>
<td>GLS Guide</td>
</tr>
<tr>
<td>character-string conversion</td>
<td>DBNLS</td>
<td>3-25</td>
</tr>
<tr>
<td>Communication Support Module: DCE-GSS</td>
<td>INFORMIXKEYTAB</td>
<td>3-49</td>
</tr>
<tr>
<td>concesm.cfg file</td>
<td>INFORMIXCONCSMCFG</td>
<td>3-47</td>
</tr>
<tr>
<td>Compiler:</td>
<td>INFORMIXC</td>
<td>3-47</td>
</tr>
<tr>
<td>multibyte characters</td>
<td>CC8BITLEVEL</td>
<td>GLS Guide</td>
</tr>
<tr>
<td>C++</td>
<td>INFORMIXCPPMAP</td>
<td>3-49</td>
</tr>
<tr>
<td>ESQL/C</td>
<td>THREADLIB</td>
<td>3-68</td>
</tr>
<tr>
<td>Topic</td>
<td>Environment Variable</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Configuration file:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>database server</td>
<td>ONCONFIG</td>
<td>3-58</td>
</tr>
<tr>
<td>ignore environment variables</td>
<td>ENVIGNORE</td>
<td>3-35</td>
</tr>
<tr>
<td>Configuration parameter:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COSERVER</td>
<td>INFORMIXSERVER</td>
<td>3-50</td>
</tr>
<tr>
<td>DBSERVERNAME</td>
<td>INFORMIXSERVER</td>
<td>3-50</td>
</tr>
<tr>
<td>DBSPACETEMP</td>
<td>DBSPACETEMP</td>
<td>3-29</td>
</tr>
<tr>
<td>DIRECTIVES</td>
<td>IFX_DIRECTIVES</td>
<td>3-40</td>
</tr>
<tr>
<td></td>
<td>IFX_EXTDIRECTIVES</td>
<td>3-41</td>
</tr>
<tr>
<td>OPCACHEMAX</td>
<td>INFORMIXOPCACHE</td>
<td>3-50</td>
</tr>
<tr>
<td>OPTCOMPIND</td>
<td>OPTCOMPIND</td>
<td>3-59</td>
</tr>
<tr>
<td>OPT_GOAL</td>
<td>OPT_GOAL</td>
<td>3-60</td>
</tr>
<tr>
<td>PDQPRORITY</td>
<td>PDQPRORITY</td>
<td>3-61</td>
</tr>
<tr>
<td>STACKSIZE</td>
<td>INFORMIXSTACKSIZE</td>
<td>3-52</td>
</tr>
<tr>
<td>Connecting</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INFORMIXCONRETRY</td>
<td>3-47</td>
</tr>
<tr>
<td></td>
<td>INFORMIXCONTIME</td>
<td>3-47</td>
</tr>
<tr>
<td></td>
<td>INFORMIXSERVER</td>
<td>3-50</td>
</tr>
<tr>
<td></td>
<td>INFORMIXSQLHOSTS</td>
<td>3-51</td>
</tr>
<tr>
<td>Data distributions</td>
<td>DBUPSPACE</td>
<td>3-33</td>
</tr>
<tr>
<td>Database locale</td>
<td>DB_LOCALE</td>
<td>GLS Guide</td>
</tr>
<tr>
<td>Database server</td>
<td>INFORMIXSERVER</td>
<td>3-50</td>
</tr>
<tr>
<td>locale for file I/O</td>
<td>SERVER_LOCALE</td>
<td>GLS Guide</td>
</tr>
<tr>
<td>configuration file</td>
<td>ONCONFIG</td>
<td>3-58</td>
</tr>
<tr>
<td>parallel sorting</td>
<td>PSORT_DBTEMP</td>
<td>3-63</td>
</tr>
<tr>
<td></td>
<td>PSORT_NPROCS</td>
<td>3-64</td>
</tr>
<tr>
<td>parallelism</td>
<td>PDQPRORITY</td>
<td>3-61</td>
</tr>
<tr>
<td>role separation</td>
<td>INF_ROLE_SEP</td>
<td>3-53</td>
</tr>
<tr>
<td>shared memory</td>
<td>INFORMIXSHMBASE</td>
<td>3-51</td>
</tr>
<tr>
<td>stacksize</td>
<td>INFORMIXSTACKSIZE</td>
<td>3-52</td>
</tr>
<tr>
<td>temporary tables</td>
<td>DBSPACETEMP</td>
<td>3-29</td>
</tr>
<tr>
<td></td>
<td>DBTEMP</td>
<td>3-30</td>
</tr>
<tr>
<td></td>
<td>PSORT_DBTEMP</td>
<td>3-63</td>
</tr>
<tr>
<td>variable-length packets</td>
<td>IFX_PAD_VARCHAR</td>
<td>3-44</td>
</tr>
<tr>
<td>Date and time values, formats</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DBCENTURY</td>
<td>3-18</td>
</tr>
<tr>
<td></td>
<td>DBDATE</td>
<td>3-20</td>
</tr>
<tr>
<td></td>
<td>GL_DATE</td>
<td>GLS Guide</td>
</tr>
<tr>
<td></td>
<td>DBTIME</td>
<td>3-31</td>
</tr>
<tr>
<td></td>
<td>GL_DATETIME</td>
<td>GLS Guide</td>
</tr>
<tr>
<td></td>
<td>IBM_XPS_PARAMS</td>
<td>3-37</td>
</tr>
<tr>
<td></td>
<td>USE_DTENV</td>
<td>IBM Informix ESQL/C Programmer’s Manual</td>
</tr>
<tr>
<td>Topic</td>
<td>Environment Variable</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>DB-Access utility</td>
<td>DBANSIWARN</td>
<td>3-17</td>
</tr>
<tr>
<td></td>
<td>DBDELMITER</td>
<td>3-22</td>
</tr>
<tr>
<td></td>
<td>DBEDIT</td>
<td>3-22</td>
</tr>
<tr>
<td></td>
<td>DBFLTMASK</td>
<td>3-22</td>
</tr>
<tr>
<td></td>
<td>DBNLS</td>
<td>3-23</td>
</tr>
<tr>
<td></td>
<td>DBPATH</td>
<td>3-26</td>
</tr>
<tr>
<td></td>
<td>FET_BUF_SIZE</td>
<td>3-36</td>
</tr>
<tr>
<td></td>
<td>INFORMIXSERVER</td>
<td>3-36</td>
</tr>
<tr>
<td></td>
<td>INFORMIXTERM</td>
<td>3-36</td>
</tr>
<tr>
<td></td>
<td>TERM</td>
<td>3-46</td>
</tr>
<tr>
<td></td>
<td>TERMCAP</td>
<td>3-46</td>
</tr>
<tr>
<td></td>
<td>TERMINFO</td>
<td>3-46</td>
</tr>
<tr>
<td>dbexport utility</td>
<td>DBDELMITER</td>
<td>3-22</td>
</tr>
<tr>
<td>dbschema utility</td>
<td>TOBIGINT</td>
<td>3-68</td>
</tr>
<tr>
<td>Delimited identifiers</td>
<td>DELIMIDENT</td>
<td>3-34</td>
</tr>
<tr>
<td>Disk space</td>
<td>DBUPSPACE</td>
<td>3-33</td>
</tr>
<tr>
<td>Editor</td>
<td>DBEDIT</td>
<td>3-22</td>
</tr>
<tr>
<td>Enterprise Replication</td>
<td>CDR_LOGDELTA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDR_PERFLOG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDR_ROUTER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDR_RMSCALEFACT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDRSITES_731</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CDRSITES_92X</td>
<td></td>
</tr>
<tr>
<td>ESQL/C: ANSI compliance</td>
<td>DBANSIWARN</td>
<td>3-17</td>
</tr>
<tr>
<td></td>
<td>INFORMIXC</td>
<td>3-47</td>
</tr>
<tr>
<td></td>
<td>DATETIME formatting</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DBTIME</td>
<td>3-31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>delimited identifiers</td>
<td>DELIMIDENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>multibyte characters</td>
<td>CLIENT_LOCALE, ESQLMF</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>multithreaded applications</td>
<td>THREADLIB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C preprocessor</td>
<td>CPFIRST</td>
</tr>
<tr>
<td>Executable programs</td>
<td>PATH</td>
<td>3-61</td>
</tr>
<tr>
<td>Fetch buffer size</td>
<td>FET_BUF_SIZE</td>
<td>3-36</td>
</tr>
<tr>
<td>Filenames: multibyte</td>
<td>GLS8BITFSYS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Files: field delimiter</td>
<td>DBDELMITER</td>
</tr>
<tr>
<td></td>
<td>Files: installation</td>
<td>INFORMIXDIR</td>
</tr>
<tr>
<td></td>
<td>Files: locale</td>
<td>CLIENT_LOCALE, DB_LOCALE, SERVER_LOCALE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Files: map for C++</td>
<td>INFORMIXCPPMAP</td>
</tr>
<tr>
<td></td>
<td>Files: message</td>
<td>DBLANG</td>
</tr>
<tr>
<td></td>
<td>Files: temporary</td>
<td>DBSPACETEMP</td>
</tr>
<tr>
<td></td>
<td>Files: temporary, for Gateways</td>
<td>DBTEMP</td>
</tr>
<tr>
<td></td>
<td>Files: temporary sorting</td>
<td>PSORT_DBTEMP</td>
</tr>
<tr>
<td>Topic</td>
<td>Environment Variable</td>
<td>Page</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------</td>
<td>------</td>
</tr>
<tr>
<td>Files: termcap, terminfo</td>
<td>INFORMIXTERM</td>
<td>3-52</td>
</tr>
<tr>
<td></td>
<td>TERM</td>
<td>3-66</td>
</tr>
<tr>
<td></td>
<td>TERMCAP</td>
<td>3-66</td>
</tr>
<tr>
<td></td>
<td>TERMINFO</td>
<td>3-67</td>
</tr>
<tr>
<td>Formats: date and time</td>
<td>DBDATE</td>
<td>3-20</td>
</tr>
<tr>
<td></td>
<td>GL_DATE</td>
<td>GLS Guide</td>
</tr>
<tr>
<td></td>
<td>DBTIME</td>
<td>3-31</td>
</tr>
<tr>
<td></td>
<td>GL_DATETIME</td>
<td>GLS Guide</td>
</tr>
<tr>
<td>Format: money</td>
<td>DBMONEY</td>
<td>3-24</td>
</tr>
<tr>
<td></td>
<td>GLS Guide</td>
<td></td>
</tr>
<tr>
<td>Gateways</td>
<td>DBTEMP</td>
<td>3-30</td>
</tr>
<tr>
<td>High-Performance Loader</td>
<td>DBONPLOAD</td>
<td>3-26</td>
</tr>
<tr>
<td></td>
<td>PLCONFIG</td>
<td>3-62</td>
</tr>
<tr>
<td></td>
<td>PLOAD_LO_PATH</td>
<td>3-65</td>
</tr>
<tr>
<td></td>
<td>PLOAD_SHMBASE</td>
<td>3-65</td>
</tr>
<tr>
<td>Identifiers: delimited</td>
<td>DELIMIDENT</td>
<td>3-34</td>
</tr>
<tr>
<td>Identifiers: longer than 18 bytes</td>
<td>IFX_LONGID</td>
<td>3-41</td>
</tr>
<tr>
<td>Identifiers: multibyte characters</td>
<td>CLIENT_LOCALE, ESQLMF</td>
<td>GLS Guide</td>
</tr>
<tr>
<td>IBM Informix Storage Manager</td>
<td>ISM_COMPRESSION</td>
<td>3-54</td>
</tr>
<tr>
<td></td>
<td>ISM_DEBUG_FILE</td>
<td>3-54</td>
</tr>
<tr>
<td></td>
<td>ISM_DEBUG_LEVEL</td>
<td>3-55</td>
</tr>
<tr>
<td></td>
<td>ISM_ENCRYPTION</td>
<td>3-55</td>
</tr>
<tr>
<td>IBM Informix Storage Manager</td>
<td>ISM_MAXLOGSIZE</td>
<td>3-55</td>
</tr>
<tr>
<td></td>
<td>ISM_MAXLOGVERS</td>
<td>3-55</td>
</tr>
<tr>
<td>Installation</td>
<td>INFORMIXDIR</td>
<td>3-49</td>
</tr>
<tr>
<td></td>
<td>PATH</td>
<td>3-61</td>
</tr>
<tr>
<td>Language environment</td>
<td>DBLANG</td>
<td>3-23</td>
</tr>
<tr>
<td></td>
<td>See also “Nondefault Locale”</td>
<td>GLS Guide</td>
</tr>
<tr>
<td>Libraries</td>
<td>LD_LIBRARY_PATH</td>
<td>3-57</td>
</tr>
<tr>
<td></td>
<td>LIBPATH</td>
<td>3-57</td>
</tr>
<tr>
<td></td>
<td>SHLIB_PATH</td>
<td>3-65</td>
</tr>
<tr>
<td>Locale</td>
<td>CLIENT_LOCALE</td>
<td>GLS Guide</td>
</tr>
<tr>
<td></td>
<td>DB_LOCALE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SERVER_LOCALE</td>
<td></td>
</tr>
<tr>
<td>Lock Mode</td>
<td>IFX_DEF_TABLE_LOCKMODE</td>
<td>3-39</td>
</tr>
<tr>
<td>Long Identifiers</td>
<td>IFX_LONGID</td>
<td>3-41</td>
</tr>
<tr>
<td>Map file for C++</td>
<td>INFORMIXCPPMAP</td>
<td>3-49</td>
</tr>
<tr>
<td>Message chaining</td>
<td>OPTMSG</td>
<td>3-59</td>
</tr>
<tr>
<td>Message files</td>
<td>DBLANG</td>
<td>3-23</td>
</tr>
<tr>
<td></td>
<td>GLS Guide</td>
<td></td>
</tr>
<tr>
<td>Money format</td>
<td>DBMONEY</td>
<td>3-24</td>
</tr>
<tr>
<td></td>
<td>GLS Guide</td>
<td></td>
</tr>
<tr>
<td>Multibyte characters</td>
<td>CLIENT_LOCALE</td>
<td>GLS Guide</td>
</tr>
<tr>
<td></td>
<td>DB_LOCALE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SERVER_LOCALE</td>
<td></td>
</tr>
<tr>
<td>Multibyte filter</td>
<td>ESQLMF</td>
<td>GLS Guide</td>
</tr>
<tr>
<td>Multithreaded applications</td>
<td>THREADLIB</td>
<td>3-68</td>
</tr>
<tr>
<td>Topic</td>
<td>Environment Variable</td>
<td>Page</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>Network</td>
<td>DBPATH</td>
<td>3-26</td>
</tr>
<tr>
<td>Nondefault locale</td>
<td>DBNLS, CLIENT_LOCALE, DB_LOCALE, SERVER_LOCALE</td>
<td>3-25, GLS Guide</td>
</tr>
<tr>
<td>ON–Bar utility</td>
<td>ISM_COMPRESSION, ISM_DEBUG_LEVEL, ISM_ENCRYPTION</td>
<td>3-54, 3-55</td>
</tr>
<tr>
<td>ONCONFIG parameters</td>
<td>See “Configuration parameter”</td>
<td>3-70</td>
</tr>
<tr>
<td>Optical Subsystem</td>
<td>INFORMIXOCACHE</td>
<td>3-50</td>
</tr>
<tr>
<td>Optimization: directives</td>
<td>IFX_DIRECTIVES, IFX_EXTDIRECTIVES</td>
<td>3-40, 3-41</td>
</tr>
<tr>
<td>Optimization: message transfers</td>
<td>OPTMSG</td>
<td>3-59</td>
</tr>
<tr>
<td>Optimization: join method</td>
<td>OPTCOMPIND</td>
<td>3-59</td>
</tr>
<tr>
<td>Optimization: performance goal</td>
<td>OPT_GOAL</td>
<td>3-60</td>
</tr>
<tr>
<td>OPTOFC feature</td>
<td>OPTOFC</td>
<td>3-60</td>
</tr>
<tr>
<td>Parameters</td>
<td>See “Configuration parameter”</td>
<td>3-70</td>
</tr>
<tr>
<td>Pathname: archecker config file</td>
<td>AC_CONFIG</td>
<td>3-14</td>
</tr>
<tr>
<td>Pathname: C compiler</td>
<td>INFORMIXC</td>
<td>3-47</td>
</tr>
<tr>
<td>Pathname: database files</td>
<td>DBPATH</td>
<td>3-26</td>
</tr>
<tr>
<td>Pathname: executable programs</td>
<td>PATH</td>
<td>3-61</td>
</tr>
<tr>
<td>Pathname: HPL sblob handles</td>
<td>PLOAD_LO_PATH</td>
<td>3-63</td>
</tr>
<tr>
<td>Pathname: installation</td>
<td>INFORMIXDIR</td>
<td>3-49</td>
</tr>
<tr>
<td>Pathname: libraries</td>
<td>LD_LIBRARY_PATH, LIBPATH, SHLIB_PATH</td>
<td>3-57, 3-57, 3-65</td>
</tr>
<tr>
<td>Pathname: message files</td>
<td>DBLANG</td>
<td>3-23, GLS Guide</td>
</tr>
<tr>
<td>Pathname: parallel sorting</td>
<td>PSORT_DBTEMP</td>
<td>3-63</td>
</tr>
<tr>
<td>Pathname: remote shell</td>
<td>DBREMOTECMD</td>
<td>3-28</td>
</tr>
<tr>
<td>Pathname: xfer_config file</td>
<td>XFER_CONFIG</td>
<td>3-68</td>
</tr>
<tr>
<td>Preserve owner name lettercase</td>
<td>ANSIOWNER</td>
<td>3-14</td>
</tr>
<tr>
<td>Printing</td>
<td>DBPRINT</td>
<td>3-28</td>
</tr>
<tr>
<td>Privileges</td>
<td>NODEFDAC</td>
<td>3-53</td>
</tr>
<tr>
<td>Query: optimization</td>
<td>IFX_DIRECTIVES, IFX_EXTDIRECTIVES</td>
<td>3-40, 3-41</td>
</tr>
<tr>
<td></td>
<td>IFMX_OPT_FACT_TABS, IFMX_OPT_NON_DIM_TABS</td>
<td>3-38, 3-39</td>
</tr>
<tr>
<td></td>
<td>OPTCOMPIND</td>
<td>3-59</td>
</tr>
<tr>
<td></td>
<td>OPT_GOAL</td>
<td>3-60</td>
</tr>
<tr>
<td></td>
<td>RTREE_COST_ADJUST_VALUE</td>
<td>3-65</td>
</tr>
<tr>
<td>Query: prioritization</td>
<td>PDQPRIOIY</td>
<td>3-61</td>
</tr>
<tr>
<td>Remote shell</td>
<td>DBREMOTECMD</td>
<td>3-28</td>
</tr>
<tr>
<td>Role separation</td>
<td>INF_ROLE_SEP</td>
<td>3-53</td>
</tr>
</tbody>
</table>
Table 3-4. Uses for Environment Variables (continued)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Environment Variable</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rolled-back transactions</td>
<td>DBACNOIGN, IFX_XASTDCOMPLIANCE_XAEND</td>
<td>3-16, 45</td>
</tr>
<tr>
<td>Routine: DATETIME formatting</td>
<td>DBTIME</td>
<td>3-31</td>
</tr>
<tr>
<td>Server</td>
<td>See &quot;Database Server&quot;</td>
<td>3-70</td>
</tr>
<tr>
<td>Server locale</td>
<td>SERVER_LOCALE</td>
<td>GLS Guide</td>
</tr>
<tr>
<td>Shared memory</td>
<td>INFORMIXSHMBASE</td>
<td>3-51</td>
</tr>
<tr>
<td></td>
<td>PLOAD_SHMBASE</td>
<td>3-63</td>
</tr>
<tr>
<td>Shell: remote</td>
<td>DBREMOTECMD</td>
<td>3-28</td>
</tr>
<tr>
<td>Shell: search path</td>
<td>PATH</td>
<td>3-61</td>
</tr>
<tr>
<td>Sorting</td>
<td>PSORT_DBTEMP</td>
<td>3-63</td>
</tr>
<tr>
<td></td>
<td>PSORT_NPROCS</td>
<td>3-64</td>
</tr>
<tr>
<td>SQL statements:caching</td>
<td>STMT_CACHE</td>
<td>3-66</td>
</tr>
<tr>
<td></td>
<td>CONNECT</td>
<td>3-42</td>
</tr>
<tr>
<td></td>
<td>INFORMIXCONTIME</td>
<td>3-50</td>
</tr>
<tr>
<td></td>
<td>INFORMIXSERVER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CREATE TEMP TABLE</td>
<td>3-29</td>
</tr>
<tr>
<td></td>
<td>DBSPACETEMP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>DESCRIBE FOR UPDATE</td>
<td>3-44</td>
</tr>
<tr>
<td></td>
<td>IFX_UPDDESC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOAD, UNLOAD</td>
<td>3-22</td>
</tr>
<tr>
<td></td>
<td>DBDELMITMER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOAD, UNLOAD</td>
<td>3-17</td>
</tr>
<tr>
<td></td>
<td>DBBLOBBUF</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SELECT INTO TEMP</td>
<td>3-29</td>
</tr>
<tr>
<td></td>
<td>DBSPACETEMP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SET PDQ_PRIORITY</td>
<td>3-61</td>
</tr>
<tr>
<td></td>
<td>PDQ_PRIORITY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SET STMT_CACHE</td>
<td>3-66</td>
</tr>
<tr>
<td></td>
<td>STMT_CACHE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UPDATE STATISTICS</td>
<td>3-33</td>
</tr>
<tr>
<td></td>
<td>DBUSPACE</td>
<td></td>
</tr>
<tr>
<td>Stacksize</td>
<td>INFORMIXSTACKSIZE</td>
<td>3-52</td>
</tr>
<tr>
<td>String search: trailing blanks</td>
<td>LIBERAL_MATCH</td>
<td>3-57</td>
</tr>
<tr>
<td>Temporary tables</td>
<td>DBSPACETEMP</td>
<td>3-29</td>
</tr>
<tr>
<td></td>
<td>DBTEMP</td>
<td>3-30</td>
</tr>
<tr>
<td></td>
<td>PSORT_DBTEMP</td>
<td>3-63</td>
</tr>
<tr>
<td>Terminal handling</td>
<td>INFORMIXTERM</td>
<td>3-52</td>
</tr>
<tr>
<td></td>
<td>TERM</td>
<td>3-66</td>
</tr>
<tr>
<td></td>
<td>TERMCAP</td>
<td>3-66</td>
</tr>
<tr>
<td></td>
<td>TERMINFO</td>
<td>3-67</td>
</tr>
<tr>
<td>Time-limited software license</td>
<td>IFX_NO_TIMELIMIT_WARNING</td>
<td>3-43</td>
</tr>
<tr>
<td>Time zone, specifying</td>
<td>IBM_XPS_PARAMS</td>
<td>3-37</td>
</tr>
<tr>
<td>Utilities: DB-Access</td>
<td>DBANSIWARN</td>
<td>3-17</td>
</tr>
<tr>
<td></td>
<td>DBDELMITMER</td>
<td>3-22</td>
</tr>
<tr>
<td></td>
<td>DBEDIT</td>
<td>3-22</td>
</tr>
<tr>
<td></td>
<td>DBFLTMASK</td>
<td>3-23</td>
</tr>
<tr>
<td></td>
<td>DBNLS</td>
<td>3-25</td>
</tr>
<tr>
<td></td>
<td>DBPATH</td>
<td>3-26</td>
</tr>
<tr>
<td></td>
<td>FET_BUF_SIZE</td>
<td>3-36</td>
</tr>
<tr>
<td></td>
<td>IFMX_HISTORY_SIZER</td>
<td>3-38</td>
</tr>
<tr>
<td></td>
<td>INFORMIXSERVER</td>
<td>3-50</td>
</tr>
<tr>
<td></td>
<td>INFORMIXTERM</td>
<td>3-52</td>
</tr>
<tr>
<td></td>
<td>TERM</td>
<td>3-66</td>
</tr>
<tr>
<td></td>
<td>TERMCAP</td>
<td>3-66</td>
</tr>
<tr>
<td></td>
<td>TERMINFO</td>
<td>3-67</td>
</tr>
</tbody>
</table>
Table 3-4. Uses for Environment Variables (continued)

<table>
<thead>
<tr>
<th>Topic</th>
<th>Environment Variable</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Utilities: dbexport</td>
<td>DBDELIMITER</td>
<td>3-22</td>
</tr>
<tr>
<td>Utilities: ON-Bar</td>
<td>ISM_COMPRESSION,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ISM_DEBUG_LEVEL,</td>
<td>3-54</td>
</tr>
<tr>
<td></td>
<td>ISM_ENCRYPTION</td>
<td>3-55</td>
</tr>
<tr>
<td>Variables: overriding</td>
<td>ENVIGNORE</td>
<td>3-35</td>
</tr>
<tr>
<td>Year values (abbreviated)</td>
<td>DBCENTURY</td>
<td>3-18</td>
</tr>
</tbody>
</table>
Appendix A. The stores_demo Database

The stores_demo database contains a set of tables that describe an imaginary business. The examples in the IBM Informix Guide to SQL: Syntax, the IBM Informix Guide to SQL: Tutorial, and other IBM Informix publications are based on this demonstration database. The stores_demo database uses the default (U.S. English) locale and is not ANSI compliant.

This appendix contains the following sections:

- The first section describes the structure of the tables in the stores_demo database. It identifies the primary key of each table, lists the name and data type of each column, and indicates whether the column has a default value or check constraint. Indexes on columns are also identified and classified as unique, allowing duplicate values.
- The second section “The stores_demo Database Map” on page A-5 shows a map of the tables in the stores_demo database and indicates the relationships among columns.
- The third section “Primary-Foreign Key Relationships” on page A-5 describes the primary-foreign key relationships among columns.
- The final section “Data in the stores_demo Database” on page A-10 lists the data contained in each table of the stores_demo database.

For information on how to create and populate the stores_demo database, see the IBM Informix DB–Access User’s Guide. For information on how to design and implement a relational database, see the IBM Informix Database Design and Implementation Guide.

Structure of the Tables

The stores_demo database contains information about a fictitious sporting-goods distributor that services stores in the western United States. This database includes the following tables:

- customer (page A-1)
- orders (page A-2)
- items (page A-2)
- stock (page A-3)
- catalog (page A-3)
- cust_calls (page A-4)
- call_type (page A-4)
- manufact (page A-4)
- state (page A-4)

Sections that follow describe each table. The unique identifying value for each table (primary key) is shaded.

The customer Table

The customer table contains information about the retail stores that place orders from the distributor. Table A-1 on page A-2 shows the columns of the customer table.
The **zipcode** column in Table A-1 is indexed and allows duplicate values.

### Table A-1. The customer Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_num</td>
<td>SERIAL(101)</td>
<td>System-generated customer number</td>
</tr>
<tr>
<td>fname</td>
<td>CHAR(15)</td>
<td>First name of store representative</td>
</tr>
<tr>
<td>lname</td>
<td>CHAR(15)</td>
<td>Last name of store representative</td>
</tr>
<tr>
<td>company</td>
<td>CHAR(20)</td>
<td>Name of store</td>
</tr>
<tr>
<td>address1</td>
<td>CHAR(20)</td>
<td>First line of store address</td>
</tr>
<tr>
<td>address2</td>
<td>CHAR(20)</td>
<td>Second line of store address</td>
</tr>
<tr>
<td>city</td>
<td>CHAR(15)</td>
<td>City</td>
</tr>
<tr>
<td>state</td>
<td>CHAR(2)</td>
<td>State (foreign key to state table)</td>
</tr>
<tr>
<td>zipcode</td>
<td>CHAR(5)</td>
<td>Zipcode</td>
</tr>
<tr>
<td>phone</td>
<td>CHAR(18)</td>
<td>Telephone number</td>
</tr>
</tbody>
</table>

### The orders Table

The **orders** table contains information about orders placed by the customers of the distributor. Table A-2 shows the columns of the **orders** table.

### Table A-2. The orders Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>order_num</td>
<td>SERIAL(1001)</td>
<td>System-generated order number</td>
</tr>
<tr>
<td>order_date</td>
<td>DATE</td>
<td>Date order entered</td>
</tr>
<tr>
<td>customer_num</td>
<td>INTEGER</td>
<td>Customer number (foreign key to customer table)</td>
</tr>
<tr>
<td>ship_instruct</td>
<td>CHAR(40)</td>
<td>Special shipping instructions</td>
</tr>
<tr>
<td>backlog</td>
<td>CHAR(1)</td>
<td>Indicates order cannot be filled because the item is backlogged: y = yes, n = no</td>
</tr>
<tr>
<td>po_num</td>
<td>CHAR(10)</td>
<td>Customer purchase order number</td>
</tr>
<tr>
<td>ship_date</td>
<td>DATE</td>
<td>Shipping date</td>
</tr>
<tr>
<td>ship_weight</td>
<td>DECIMAL(8,2)</td>
<td>Shipping weight</td>
</tr>
<tr>
<td>ship_charge</td>
<td>MONEY(6)</td>
<td>Shipping charge</td>
</tr>
<tr>
<td>paid_date</td>
<td>DATE</td>
<td>Date order paid</td>
</tr>
</tbody>
</table>

### The items Table

An order can include one or more items. One row exists in the **items** table for each item in an order. Table A-3 on page A-3 shows the columns of the **items** table.
Table A-3. The items Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item_num</td>
<td>SMALLINT</td>
<td>Sequentially assigned item number for an order</td>
</tr>
<tr>
<td>order_num</td>
<td>INTEGER</td>
<td>Order number (foreign key to orders table)</td>
</tr>
<tr>
<td>stock_num</td>
<td>SMALLINT</td>
<td>Stock number for item (foreign key to stock table)</td>
</tr>
<tr>
<td>manu_code</td>
<td>CHAR(3)</td>
<td>Manufacturer code for item ordered (foreign key to manufact table)</td>
</tr>
<tr>
<td>quantity</td>
<td>SMALLINT</td>
<td>Quantity ordered (value must be &gt; 1)</td>
</tr>
<tr>
<td>total_price</td>
<td>MONEY(8)</td>
<td>Quantity ordered * unit price = total price of item</td>
</tr>
</tbody>
</table>

The stock Table

The distributor carries 41 types of sporting goods from various manufacturers. More than one manufacturer can supply an item. For example, the distributor offers racing goggles from two manufacturers and running shoes from six manufacturers.

The stock table is a catalog of the items sold by the distributor. Table A-4 shows the columns of the stock table.

Table A-4. The stock Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stock_num</td>
<td>SMALLINT</td>
<td>Stock number that identifies type of item</td>
</tr>
<tr>
<td>manu_code</td>
<td>CHAR(3)</td>
<td>Manufacturer code (foreign key to manufact table)</td>
</tr>
<tr>
<td>description</td>
<td>CHAR(15)</td>
<td>Description of item</td>
</tr>
<tr>
<td>unit_price</td>
<td>MONEY(6,2)</td>
<td>Unit price</td>
</tr>
<tr>
<td>unit</td>
<td>CHAR(4)</td>
<td>Unit by which item is ordered:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Each</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Pair</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Case</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Box</td>
</tr>
<tr>
<td>unit_descr</td>
<td>CHAR(15)</td>
<td>Description of unit</td>
</tr>
</tbody>
</table>

The catalog Table

The catalog table describes each item in stock. Retail stores use this table when placing orders with the distributor. Table A-5 shows the columns of the catalog table.

Table A-5. The catalog Table

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalog_num</td>
<td>SERIAL(10001)</td>
<td>System-generated catalog number</td>
</tr>
<tr>
<td>stock_num</td>
<td>SMALLINT</td>
<td>Distributor stock number (foreign key to stock table)</td>
</tr>
<tr>
<td>manu_code</td>
<td>CHAR(3)</td>
<td>Manufacturer code (foreign key to manufact table)</td>
</tr>
<tr>
<td>cat_descr</td>
<td>TEXT</td>
<td>Description of item</td>
</tr>
<tr>
<td>cat_picture</td>
<td>BYTE</td>
<td>Picture of item (binary data)</td>
</tr>
<tr>
<td>cat_advert</td>
<td>VARCHAR(255, 65)</td>
<td>Tag line underneath picture</td>
</tr>
</tbody>
</table>
The cust_calls Table

All customer calls for information on orders, shipments, or complaints are logged. The cust_calls table contains information about these types of customer calls. Table A-6 shows the columns of the cust_calls table.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_num</td>
<td>INTEGER</td>
<td>Customer number (foreign key to customer table)</td>
</tr>
<tr>
<td>call_dtime</td>
<td>DATETIME YEAR TO MINUTE</td>
<td>Date and time when call was received</td>
</tr>
<tr>
<td>user_id</td>
<td>CHAR(18)</td>
<td>Name of person logging call (default is user login name)</td>
</tr>
<tr>
<td>call_code</td>
<td>CHAR(1)</td>
<td>Type of call (foreign key to call_type table)</td>
</tr>
<tr>
<td>call_descr</td>
<td>CHAR(240)</td>
<td>Description of call</td>
</tr>
<tr>
<td>res_dtime</td>
<td>DATETIME YEAR TO MINUTE</td>
<td>Date and time when call was resolved</td>
</tr>
<tr>
<td>res_descr</td>
<td>CHAR(240)</td>
<td>Description of how call was resolved</td>
</tr>
</tbody>
</table>

The call_type Table

The call codes associated with customer calls are stored in the call_type table. Table A-7 shows the columns of the call_type table.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>call_code</td>
<td>CHAR(1)</td>
<td>Call code</td>
</tr>
<tr>
<td>code_descr</td>
<td>CHAR(30)</td>
<td>Description of call type</td>
</tr>
</tbody>
</table>

The manufact Table

Information about the nine manufacturers whose sporting goods are handled by the distributor is stored in the manufact table. Table A-8 shows the columns of the manufact table.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>manu_code</td>
<td>CHAR(3)</td>
<td>Manufacturer code</td>
</tr>
<tr>
<td>manu_name</td>
<td>CHAR(15)</td>
<td>Name of manufacturer</td>
</tr>
<tr>
<td>lead_time</td>
<td>INTERVAL DAY(3) TO DAY</td>
<td>Lead time for shipment of orders</td>
</tr>
</tbody>
</table>

The state Table

The state table contains the names and postal abbreviations for the 50 states of the United States. Table A-9 shows the columns of the state table.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>CHAR(2)</td>
<td>State code</td>
</tr>
<tr>
<td>sname</td>
<td>CHAR(15)</td>
<td>State name</td>
</tr>
</tbody>
</table>
The stores_demo Database Map

Figure A-1 displays the joins in the stores_demo database. The gray shading that connects a column in one table to a column with the same name in another table indicates the relationships, or joins, between tables.

Primary-Foreign Key Relationships

The tables of the stores_demo database are linked by the primary-foreign key relationships that Figure A-1 shows and are identified in this section. This type of relationship is called a referential constraint because a foreign key in one table references the primary key in another table. Figure A-2 on page A-6 through
The customer and orders Tables

The customer table contains a customer_num column that holds a number that identifies a customer and columns for the customer name, company, address, and telephone number. For example, the row with information about Anthony Higgins contains the number 104 in the customer_num column. The orders table also contains a customer_num column that stores the number of the customer who placed a particular order. In the orders table, the customer_num column is a foreign key that references the customer_num column in the customer table. Figure A-2 shows this relationship.

The orders and items Tables

The orders and items tables are linked by an order_num column that contains an identification number for each order. If an order includes several items, the same order number appears in several rows of the items table. In the items table, the order_num column is a foreign key that references the order_num column in the orders table. Figure A-3 on page A-7 shows this relationship.
The items and stock Tables

The items table and the stock table are joined by two columns: the stock_num column, which stores a stock number for an item, and the manu_code column, which stores a code that identifies the manufacturer. You need both the stock number and the manufacturer code to uniquely identify an item. For example, the item with the stock number 1 and the manufacturer code HRO is a Hero baseball glove; the item with the stock number 1 and the manufacturer code HSK is a Husky baseball glove.

The same stock number and manufacturer code can appear in more than one row of the items table, if the same item belongs to separate orders. In the items table, the stock_num and manu_code columns are foreign keys that reference the stock_num and manu_code columns in the stock table. Figure A-4 shows this relationship.

Figure A-4. Tables That the stock_num and manu_code Columns Join
The stock and catalog Tables

The stock table and catalog table are joined by two columns: the stock_num column, which stores a stock number for an item, and the manu_code column, which stores a code that identifies the manufacturer. You need both columns to uniquely identify an item. In the catalog table, the stock_num and manu_code columns are foreign keys that reference the stock_num and manu_code columns in the stock table. Figure A-5 shows this relationship.

```
stock Table (detail)
<table>
<thead>
<tr>
<th>stock_num</th>
<th>manu_code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HRO</td>
<td>baseball gloves</td>
</tr>
<tr>
<td>1</td>
<td>HSK</td>
<td>baseball gloves</td>
</tr>
<tr>
<td>1</td>
<td>SMT</td>
<td>baseball gloves</td>
</tr>
</tbody>
</table>
```

```
catalog Table (detail)
<table>
<thead>
<tr>
<th>catalog_num</th>
<th>stock_num</th>
<th>manu_code</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>1</td>
<td>HRO</td>
</tr>
<tr>
<td>10002</td>
<td>1</td>
<td>HSK</td>
</tr>
<tr>
<td>10003</td>
<td>1</td>
<td>SMT</td>
</tr>
<tr>
<td>10004</td>
<td>2</td>
<td>HRO</td>
</tr>
</tbody>
</table>
```

Figure A-5. Tables That the stock_num and manu_code Columns Join

The stock and manufact Tables

The stock table and the manufact table are joined by the manu_code column. The same manufacturer code can appear in more than one row of the stock table if the manufacturer produces more than one piece of equipment. In the stock table, the manu_code column is a foreign key that references the manu_code column in the manufact table. Figure A-6 shows this relationship.

```
stock Table (detail)
<table>
<thead>
<tr>
<th>stock_num</th>
<th>manu_code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HRO</td>
<td>baseball gloves</td>
</tr>
<tr>
<td>1</td>
<td>HSK</td>
<td>baseball gloves</td>
</tr>
<tr>
<td>1</td>
<td>SMT</td>
<td>baseball gloves</td>
</tr>
</tbody>
</table>
```

```
manufact Table (detail)
<table>
<thead>
<tr>
<th>manu_code</th>
<th>manu_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRG</td>
<td>Norge</td>
</tr>
<tr>
<td>HSK</td>
<td>Husky</td>
</tr>
<tr>
<td>HRO</td>
<td>Hero</td>
</tr>
</tbody>
</table>
```

Figure A-6. Tables That the manu_code Column Joins

The cust_calls and customer Tables

The cust_calls table and the customer table are joined by the customer_num column. The same customer number can appear in more than one row of the cust_calls table if the customer calls the distributor more than once with a problem or question. In the cust_calls table, the customer_num column is a foreign key that references the customer_num column in the customer table. Figure A-7 on page A-9 shows this relationship.
The call_type and cust_calls Tables

The call_type and cust_calls tables are joined by the call_code column. The same call code can appear in more than one row of the cust_calls table because many customers can have the same type of problem. In the cust_calls table, the call_code column is a foreign key that references the call_code column in the call_type table. Figure A-8 shows this relationship.

The state and customer Tables

The state table and the customer table are joined by a column that contains the state code. This column is called code in the state table and state in the customer table. If several customers live in the same state, the same state code appears in several rows of the table. In the customer table, the state column is a foreign key that references the code column in the state table. Figure A-9 on page A-10 shows this relationship.
### Data in the stores_demo Database

The following tables display the data in the `stores_demo` database.

#### customer Table

<table>
<thead>
<tr>
<th>customer_num</th>
<th>fname</th>
<th>lname</th>
<th>company</th>
<th>address1</th>
<th>address2</th>
<th>city</th>
<th>state</th>
<th>zipcode</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Ludwig</td>
<td>Pauli</td>
<td>All Sports Supplies</td>
<td>213 Erstwild Court</td>
<td></td>
<td>Sunnyvale</td>
<td>CA</td>
<td>94086</td>
<td>408-789-8075</td>
</tr>
<tr>
<td>102</td>
<td>Carole</td>
<td>Sadler</td>
<td>Sports Spot</td>
<td>785 Geary Street</td>
<td></td>
<td>San Francisco</td>
<td>CA</td>
<td>94117</td>
<td>415-822-1289</td>
</tr>
<tr>
<td>103</td>
<td>Philip</td>
<td>Currie</td>
<td>Phil's Sports</td>
<td>654 Poplar</td>
<td>P. O. Box 3498</td>
<td>Palo Alto</td>
<td>CA</td>
<td>94303</td>
<td>650-328-4543</td>
</tr>
<tr>
<td>104</td>
<td>Anthony</td>
<td>Higgins</td>
<td>Play Ball!</td>
<td>East Shopping Center</td>
<td>422 Bay Road</td>
<td>Redwood City</td>
<td>CA</td>
<td>94026</td>
<td>650-368-1100</td>
</tr>
<tr>
<td>105</td>
<td>Raymond</td>
<td>Vector</td>
<td>Los Altos Sports</td>
<td>1899 La Loma Drive</td>
<td></td>
<td>Los Altos</td>
<td>CA</td>
<td>94022</td>
<td>650-776-3249</td>
</tr>
<tr>
<td>106</td>
<td>George</td>
<td>Watson</td>
<td>Watson &amp; Son</td>
<td>1143 Carver Place</td>
<td></td>
<td>Mountain View</td>
<td>CA</td>
<td>94063</td>
<td>650-389-8789</td>
</tr>
<tr>
<td>107</td>
<td>Charles</td>
<td>Ream</td>
<td>Athletic Supplies</td>
<td>41 Jordan Avenue</td>
<td></td>
<td>Palo Alto</td>
<td>CA</td>
<td>94304</td>
<td>650-356-9876</td>
</tr>
<tr>
<td>108</td>
<td>Donald</td>
<td>Quinn</td>
<td>Quinn’s Sports</td>
<td>587 Alvarado</td>
<td></td>
<td>Redwood City</td>
<td>CA</td>
<td>94063</td>
<td>650-544-8729</td>
</tr>
<tr>
<td>customer_num</td>
<td>fname</td>
<td>lname</td>
<td>company</td>
<td>address1</td>
<td>address2</td>
<td>city</td>
<td>state</td>
<td>zipcode</td>
<td>phone</td>
</tr>
<tr>
<td>-------------</td>
<td>--------</td>
<td>----------</td>
<td>------------------</td>
<td>----------------</td>
<td>-------------------------</td>
<td>----------</td>
<td>-------</td>
<td>---------</td>
<td>------------------</td>
</tr>
<tr>
<td>109</td>
<td>Jane</td>
<td>Miller</td>
<td>Sport Stuff</td>
<td>Mayfair Mart</td>
<td>7345 Ross Blvd.</td>
<td>Sunnyvale</td>
<td>CA</td>
<td>94086</td>
<td>408-723-8789</td>
</tr>
<tr>
<td>110</td>
<td>Roy</td>
<td>Jaeger</td>
<td>AA Athletics</td>
<td>520 Topaz Way</td>
<td>Redwood City</td>
<td>CA</td>
<td>94062</td>
<td>650-743-3611</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>Frances</td>
<td>Keyes</td>
<td>Sports Center</td>
<td>3199 Sterling Court</td>
<td>Sunnyvale</td>
<td>CA</td>
<td>94085</td>
<td>408-277-7245</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>Margaret</td>
<td>Lawson</td>
<td>Runners &amp; Others</td>
<td>234 Wyandotte Way</td>
<td>Los Altos</td>
<td>CA</td>
<td>94022</td>
<td>650-887-7235</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>Lana</td>
<td>Beatty</td>
<td>Sportstown</td>
<td>654 Oak Grove</td>
<td>Menlo Park</td>
<td>CA</td>
<td>94025</td>
<td>650-356-9982</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>Frank</td>
<td>Albertson</td>
<td>Sporting Place</td>
<td>947 Waverly Place</td>
<td>Redwood City</td>
<td>CA</td>
<td>94062</td>
<td>650-886-6677</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>Alfred</td>
<td>Grant</td>
<td>Gold Medal Sports</td>
<td>776 Gary Avenue</td>
<td>Menlo Park</td>
<td>CA</td>
<td>94025</td>
<td>650-356-1123</td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>Jean</td>
<td>Parmelee</td>
<td>Olympic City</td>
<td>1104 Spinosa Drive</td>
<td>Mountain View</td>
<td>CA</td>
<td>94040</td>
<td>650-534-8822</td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>Arnold</td>
<td>Sipes</td>
<td>Kids Korner</td>
<td>850 Lytton Court</td>
<td>Redwood City</td>
<td>CA</td>
<td>94063</td>
<td>650-245-4578</td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>Dick</td>
<td>Baxter</td>
<td>Blue Ribbon Sports</td>
<td>5427 College</td>
<td>Oakland</td>
<td>CA</td>
<td>94609</td>
<td>650-655-0011</td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>Bob</td>
<td>Shorter</td>
<td>The Triathletes Club</td>
<td>2405 Kings Highway</td>
<td>Cherry Hill</td>
<td>NJ</td>
<td>08002</td>
<td>609-663-6079</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>Fred</td>
<td>Jewell</td>
<td>Century Pro Shop</td>
<td>6627 N. 17th Way</td>
<td>Phoenix</td>
<td>AZ</td>
<td>85016</td>
<td>602-265-8754</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>Jason</td>
<td>Wallack</td>
<td>City Sports</td>
<td>Lake Biltmore Mall</td>
<td>Wilmington</td>
<td>DE</td>
<td>19898</td>
<td>302-366-7511</td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>Cathy</td>
<td>O’Brien</td>
<td>The Sporting Life</td>
<td>543 Nassau Street</td>
<td>Princeton</td>
<td>NJ</td>
<td>08540</td>
<td>609-342-0054</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>Marvin</td>
<td>Hanlon</td>
<td>Bay Sports</td>
<td>10100 Bay Meadows Road</td>
<td>Jacksonville</td>
<td>FL</td>
<td>32256</td>
<td>904-823-4239</td>
<td></td>
</tr>
</tbody>
</table>

Appendix A. The stores_demo Database A-11
<table>
<thead>
<tr>
<th>customer_num</th>
<th>fname</th>
<th>lname</th>
<th>company</th>
<th>address1</th>
<th>address2</th>
<th>city</th>
<th>state</th>
<th>zipcode</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>124</td>
<td>Chris</td>
<td>Putnum</td>
<td>Putnum’s Putters</td>
<td>4715 S.E. Adams Blvd</td>
<td>Suite 909C</td>
<td>Bartlesville OK</td>
<td></td>
<td>74006</td>
<td>918-355-2074</td>
</tr>
<tr>
<td>125</td>
<td>James</td>
<td>Henry</td>
<td>Total Fitness Sports</td>
<td>1450 Commonwealth Avenue</td>
<td></td>
<td>Brighton MA</td>
<td></td>
<td>02135</td>
<td>617-232-4159</td>
</tr>
<tr>
<td>126</td>
<td>Eileen</td>
<td>Neelie</td>
<td>Neelie’s Discount Sports</td>
<td>2539 South Utica Street</td>
<td>Denver CO</td>
<td></td>
<td></td>
<td>80219</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>Kim</td>
<td>Satifer</td>
<td>Big Blue Bike Shop</td>
<td>Blue Island Square</td>
<td>12222 Gregory Street</td>
<td>Blue Island NY</td>
<td></td>
<td>60406</td>
<td>312-944-5691</td>
</tr>
<tr>
<td>128</td>
<td>Frank</td>
<td>Lessor</td>
<td>Phoenix University Athletic Department</td>
<td>1817 N. Thomas Road</td>
<td>Phoenix AZ</td>
<td></td>
<td></td>
<td>85008</td>
<td>602-533-1817</td>
</tr>
</tbody>
</table>

**items Table**

<table>
<thead>
<tr>
<th>item_num</th>
<th>order_num</th>
<th>stock_num</th>
<th>manu_code</th>
<th>quantity</th>
<th>total_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1001</td>
<td>1</td>
<td>HRO</td>
<td>1</td>
<td>250.00</td>
</tr>
<tr>
<td>1</td>
<td>1002</td>
<td>4</td>
<td>HSK</td>
<td>1</td>
<td>960.00</td>
</tr>
<tr>
<td>2</td>
<td>1002</td>
<td>3</td>
<td>HSK</td>
<td>1</td>
<td>240.00</td>
</tr>
<tr>
<td>1</td>
<td>1003</td>
<td>9</td>
<td>ANZ</td>
<td>1</td>
<td>20.00</td>
</tr>
<tr>
<td>2</td>
<td>1003</td>
<td>8</td>
<td>ANZ</td>
<td>1</td>
<td>840.00</td>
</tr>
<tr>
<td>3</td>
<td>1003</td>
<td>5</td>
<td>ANZ</td>
<td>5</td>
<td>99.00</td>
</tr>
<tr>
<td>1</td>
<td>1004</td>
<td>1</td>
<td>HRO</td>
<td>1</td>
<td>250.00</td>
</tr>
<tr>
<td>2</td>
<td>1004</td>
<td>2</td>
<td>HRO</td>
<td>1</td>
<td>126.00</td>
</tr>
<tr>
<td>3</td>
<td>1004</td>
<td>3</td>
<td>HSK</td>
<td>1</td>
<td>240.00</td>
</tr>
<tr>
<td>4</td>
<td>1004</td>
<td>1</td>
<td>HSK</td>
<td>1</td>
<td>800.00</td>
</tr>
<tr>
<td>1</td>
<td>1005</td>
<td>5</td>
<td>NRG</td>
<td>10</td>
<td>280.00</td>
</tr>
<tr>
<td>2</td>
<td>1005</td>
<td>5</td>
<td>ANZ</td>
<td>10</td>
<td>198.00</td>
</tr>
<tr>
<td>3</td>
<td>1005</td>
<td>6</td>
<td>SMT</td>
<td>1</td>
<td>36.00</td>
</tr>
<tr>
<td>4</td>
<td>1005</td>
<td>6</td>
<td>ANZ</td>
<td>1</td>
<td>48.00</td>
</tr>
<tr>
<td>1</td>
<td>1006</td>
<td>5</td>
<td>SMT</td>
<td>5</td>
<td>125.00</td>
</tr>
<tr>
<td>2</td>
<td>1006</td>
<td>5</td>
<td>NRG</td>
<td>5</td>
<td>140.00</td>
</tr>
<tr>
<td>3</td>
<td>1006</td>
<td>5</td>
<td>ANZ</td>
<td>5</td>
<td>99.00</td>
</tr>
<tr>
<td>4</td>
<td>1006</td>
<td>6</td>
<td>SMT</td>
<td>1</td>
<td>36.00</td>
</tr>
<tr>
<td>5</td>
<td>1006</td>
<td>6</td>
<td>ANZ</td>
<td>1</td>
<td>48.00</td>
</tr>
<tr>
<td>1</td>
<td>1007</td>
<td>1</td>
<td>HRO</td>
<td>1</td>
<td>250.00</td>
</tr>
<tr>
<td>2</td>
<td>1007</td>
<td>2</td>
<td>HRO</td>
<td>1</td>
<td>126.00</td>
</tr>
<tr>
<td>3</td>
<td>1007</td>
<td>3</td>
<td>HSK</td>
<td>1</td>
<td>240.00</td>
</tr>
<tr>
<td>item_num</td>
<td>order_num</td>
<td>stock_num</td>
<td>manu_code</td>
<td>quantity</td>
<td>total_price</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>4</td>
<td>1007</td>
<td>4</td>
<td>HRO</td>
<td>1</td>
<td>480.00</td>
</tr>
<tr>
<td>5</td>
<td>1007</td>
<td>7</td>
<td>HRO</td>
<td>1</td>
<td>600.00</td>
</tr>
<tr>
<td>1</td>
<td>1008</td>
<td>8</td>
<td>ANZ</td>
<td>1</td>
<td>840.00</td>
</tr>
<tr>
<td>2</td>
<td>1008</td>
<td>9</td>
<td>ANZ</td>
<td>5</td>
<td>100.00</td>
</tr>
<tr>
<td>1</td>
<td>1009</td>
<td>1</td>
<td>SMT</td>
<td>1</td>
<td>450.00</td>
</tr>
<tr>
<td>1</td>
<td>1010</td>
<td>6</td>
<td>SMT</td>
<td>1</td>
<td>36.00</td>
</tr>
<tr>
<td>2</td>
<td>1010</td>
<td>6</td>
<td>ANZ</td>
<td>1</td>
<td>48.00</td>
</tr>
<tr>
<td>1</td>
<td>1011</td>
<td>5</td>
<td>ANZ</td>
<td>5</td>
<td>99.00</td>
</tr>
<tr>
<td>1</td>
<td>1012</td>
<td>8</td>
<td>ANZ</td>
<td>1</td>
<td>840.00</td>
</tr>
<tr>
<td>2</td>
<td>1012</td>
<td>9</td>
<td>ANZ</td>
<td>10</td>
<td>200.00</td>
</tr>
<tr>
<td>1</td>
<td>1013</td>
<td>5</td>
<td>ANZ</td>
<td>1</td>
<td>19.80</td>
</tr>
<tr>
<td>2</td>
<td>1013</td>
<td>6</td>
<td>SMT</td>
<td>1</td>
<td>36.00</td>
</tr>
<tr>
<td>3</td>
<td>1013</td>
<td>6</td>
<td>ANZ</td>
<td>1</td>
<td>48.00</td>
</tr>
<tr>
<td>4</td>
<td>1013</td>
<td>9</td>
<td>ANZ</td>
<td>2</td>
<td>40.00</td>
</tr>
<tr>
<td>1</td>
<td>1014</td>
<td>4</td>
<td>HSK</td>
<td>1</td>
<td>960.00</td>
</tr>
<tr>
<td>2</td>
<td>1014</td>
<td>4</td>
<td>HRO</td>
<td>1</td>
<td>480.00</td>
</tr>
<tr>
<td>1</td>
<td>1015</td>
<td>1</td>
<td>SMT</td>
<td>1</td>
<td>450.00</td>
</tr>
<tr>
<td>1</td>
<td>1016</td>
<td>101</td>
<td>SHM</td>
<td>2</td>
<td>136.00</td>
</tr>
<tr>
<td>2</td>
<td>1016</td>
<td>109</td>
<td>PRC</td>
<td>3</td>
<td>90.00</td>
</tr>
<tr>
<td>3</td>
<td>1016</td>
<td>110</td>
<td>HSK</td>
<td>1</td>
<td>308.00</td>
</tr>
<tr>
<td>4</td>
<td>1016</td>
<td>114</td>
<td>PRC</td>
<td>1</td>
<td>120.00</td>
</tr>
<tr>
<td>1</td>
<td>1017</td>
<td>201</td>
<td>NKL</td>
<td>4</td>
<td>150.00</td>
</tr>
<tr>
<td>2</td>
<td>1017</td>
<td>202</td>
<td>KAR</td>
<td>1</td>
<td>230.00</td>
</tr>
<tr>
<td>3</td>
<td>1017</td>
<td>301</td>
<td>SHM</td>
<td>2</td>
<td>204.00</td>
</tr>
<tr>
<td>1</td>
<td>1018</td>
<td>307</td>
<td>PRC</td>
<td>2</td>
<td>500.00</td>
</tr>
<tr>
<td>2</td>
<td>1018</td>
<td>302</td>
<td>KAR</td>
<td>3</td>
<td>15.00</td>
</tr>
<tr>
<td>3</td>
<td>1018</td>
<td>110</td>
<td>PRC</td>
<td>1</td>
<td>236.00</td>
</tr>
<tr>
<td>4</td>
<td>1018</td>
<td>5</td>
<td>SMT</td>
<td>4</td>
<td>100.00</td>
</tr>
<tr>
<td>5</td>
<td>1018</td>
<td>304</td>
<td>HRO</td>
<td>1</td>
<td>280.00</td>
</tr>
<tr>
<td>1</td>
<td>1019</td>
<td>111</td>
<td>SHM</td>
<td>3</td>
<td>1499.97</td>
</tr>
<tr>
<td>1</td>
<td>1020</td>
<td>204</td>
<td>KAR</td>
<td>2</td>
<td>90.00</td>
</tr>
<tr>
<td>2</td>
<td>1020</td>
<td>301</td>
<td>KAR</td>
<td>4</td>
<td>348.00</td>
</tr>
<tr>
<td>1</td>
<td>1021</td>
<td>201</td>
<td>NKL</td>
<td>2</td>
<td>75.00</td>
</tr>
<tr>
<td>2</td>
<td>1021</td>
<td>201</td>
<td>ANZ</td>
<td>3</td>
<td>225.00</td>
</tr>
<tr>
<td>3</td>
<td>1021</td>
<td>202</td>
<td>KAR</td>
<td>3</td>
<td>690.00</td>
</tr>
<tr>
<td>4</td>
<td>1021</td>
<td>205</td>
<td>ANZ</td>
<td>2</td>
<td>624.00</td>
</tr>
<tr>
<td>1</td>
<td>1022</td>
<td>309</td>
<td>HRO</td>
<td>1</td>
<td>40.00</td>
</tr>
<tr>
<td>2</td>
<td>1022</td>
<td>303</td>
<td>PRC</td>
<td>2</td>
<td>96.00</td>
</tr>
<tr>
<td>3</td>
<td>1022</td>
<td>6</td>
<td>ANZ</td>
<td>2</td>
<td>96.00</td>
</tr>
<tr>
<td>1</td>
<td>1023</td>
<td>103</td>
<td>PRC</td>
<td>2</td>
<td>40.00</td>
</tr>
<tr>
<td>2</td>
<td>1023</td>
<td>104</td>
<td>PRC</td>
<td>2</td>
<td>116.00</td>
</tr>
</tbody>
</table>
### item Table

<table>
<thead>
<tr>
<th>item_num</th>
<th>order_num</th>
<th>stock_num</th>
<th>manu_code</th>
<th>quantity</th>
<th>total_price</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1023</td>
<td>105</td>
<td>SHM</td>
<td>1</td>
<td>80.00</td>
</tr>
<tr>
<td>4</td>
<td>1023</td>
<td>110</td>
<td>SHM</td>
<td>1</td>
<td>228.00</td>
</tr>
<tr>
<td>5</td>
<td>1023</td>
<td>304</td>
<td>ANZ</td>
<td>1</td>
<td>170.00</td>
</tr>
<tr>
<td>6</td>
<td>1023</td>
<td>306</td>
<td>SHM</td>
<td>1</td>
<td>190.00</td>
</tr>
</tbody>
</table>

### call_type Table

<table>
<thead>
<tr>
<th>call_code</th>
<th>code_descr</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>billing error</td>
</tr>
<tr>
<td>D</td>
<td>damaged goods</td>
</tr>
<tr>
<td>I</td>
<td>incorrect merchandise sent</td>
</tr>
<tr>
<td>L</td>
<td>late shipment</td>
</tr>
<tr>
<td>O</td>
<td>other</td>
</tr>
</tbody>
</table>

### orders Table

<table>
<thead>
<tr>
<th>order_num</th>
<th>order_date</th>
<th>customer_num</th>
<th>ship_instruct</th>
<th>backlog</th>
<th>po_num</th>
<th>ship_date</th>
<th>ship_weight</th>
<th>ship_charge</th>
<th>paid_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>05/20/1998</td>
<td>104</td>
<td>express</td>
<td>n</td>
<td>B77836</td>
<td>06/01/1998</td>
<td>20.40</td>
<td>10.00</td>
<td>07/22/1998</td>
</tr>
<tr>
<td>1002</td>
<td>05/21/1998</td>
<td>101</td>
<td>PO on box; deliver back door only</td>
<td>n</td>
<td>9270</td>
<td>05/26/1998</td>
<td>50.60</td>
<td>15.30</td>
<td>06/03/1998</td>
</tr>
<tr>
<td>1003</td>
<td>05/22/1998</td>
<td>104</td>
<td>express</td>
<td>n</td>
<td>B77890</td>
<td>05/23/1998</td>
<td>35.60</td>
<td>10.80</td>
<td>06/14/1998</td>
</tr>
<tr>
<td>1004</td>
<td>05/22/1998</td>
<td>106</td>
<td>ring bell twice</td>
<td>y</td>
<td>8006</td>
<td>05/30/1998</td>
<td>95.80</td>
<td>19.20</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td>05/30/1998</td>
<td>112</td>
<td>after 10AM</td>
<td>y</td>
<td>Q13557</td>
<td>06/09/1998</td>
<td>70.80</td>
<td>14.20</td>
<td></td>
</tr>
<tr>
<td>1007</td>
<td>05/31/1998</td>
<td>117</td>
<td></td>
<td>n</td>
<td>278693</td>
<td>06/05/1998</td>
<td>125.90</td>
<td>25.20</td>
<td></td>
</tr>
<tr>
<td>1009</td>
<td>06/14/1998</td>
<td>111</td>
<td>door next to grocery</td>
<td>n</td>
<td>4745</td>
<td>06/21/1998</td>
<td>20.40</td>
<td>10.00</td>
<td>08/21/1998</td>
</tr>
<tr>
<td>1010</td>
<td>06/17/1998</td>
<td>115</td>
<td>deliver 776 King St. if no answer</td>
<td>n</td>
<td>429Q</td>
<td>06/29/1998</td>
<td>40.60</td>
<td>12.30</td>
<td>08/22/1998</td>
</tr>
<tr>
<td>1011</td>
<td>06/18/1998</td>
<td>104</td>
<td>express</td>
<td>n</td>
<td>B77897</td>
<td>07/03/1998</td>
<td>10.40</td>
<td>5.00</td>
<td>08/29/1998</td>
</tr>
<tr>
<td>1012</td>
<td>06/18/1998</td>
<td>117</td>
<td></td>
<td>n</td>
<td>278701</td>
<td>06/29/1998</td>
<td>70.80</td>
<td>14.20</td>
<td></td>
</tr>
<tr>
<td>1016</td>
<td>06/29/1998</td>
<td>119</td>
<td>delivery entrance off Camp St.</td>
<td>n</td>
<td>PC6782</td>
<td>07/12/1998</td>
<td>35.00</td>
<td>11.80</td>
<td></td>
</tr>
<tr>
<td>order_num</td>
<td>order_date</td>
<td>customer_num</td>
<td>ship_instruct</td>
<td>backlog</td>
<td>po_num</td>
<td>ship_date</td>
<td>ship_weight</td>
<td>ship_charge</td>
<td>paid_date</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>--------------</td>
<td>---------------</td>
<td>---------</td>
<td>--------</td>
<td>-----------</td>
<td>-------------</td>
<td>-------------</td>
<td>-----------</td>
</tr>
<tr>
<td>1017</td>
<td>07/09/1998</td>
<td>120</td>
<td>North side of clubhouse</td>
<td>n</td>
<td>DM3543</td>
<td>07/13/1998</td>
<td>60.00</td>
<td>18.00</td>
<td></td>
</tr>
<tr>
<td>1018</td>
<td>07/10/1998</td>
<td>121</td>
<td>SW corner of Biltmore Mall</td>
<td>n</td>
<td>S22942</td>
<td>07/13/1998</td>
<td>70.50</td>
<td>20.00</td>
<td>08/06/1998</td>
</tr>
<tr>
<td>1019</td>
<td>07/11/1998</td>
<td>122</td>
<td>closed til noon Mondays</td>
<td>n</td>
<td>Z55709</td>
<td>07/16/1998</td>
<td>90.00</td>
<td>23.00</td>
<td>08/06/1998</td>
</tr>
<tr>
<td>1022</td>
<td>07/24/1998</td>
<td>126</td>
<td>express</td>
<td>n</td>
<td>W9925</td>
<td>07/30/1998</td>
<td>15.00</td>
<td>13.00</td>
<td>09/02/1998</td>
</tr>
<tr>
<td>1023</td>
<td>07/24/1998</td>
<td>127</td>
<td>no deliveries after 3 p.m.</td>
<td>n</td>
<td>KF2961</td>
<td>07/30/1998</td>
<td>60.00</td>
<td>18.00</td>
<td>08/22/1998</td>
</tr>
</tbody>
</table>

**stock Table**

<table>
<thead>
<tr>
<th>stock_num</th>
<th>manu_code</th>
<th>description</th>
<th>unit_rice</th>
<th>unit</th>
<th>unit_descr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HRO</td>
<td>baseball gloves</td>
<td>250.00</td>
<td>case</td>
<td>10 gloves/case</td>
</tr>
<tr>
<td>1</td>
<td>HSK</td>
<td>baseball gloves</td>
<td>800.00</td>
<td>case</td>
<td>10 gloves/case</td>
</tr>
<tr>
<td>1</td>
<td>SMT</td>
<td>baseball gloves</td>
<td>450.00</td>
<td>case</td>
<td>10 gloves/case</td>
</tr>
<tr>
<td>2</td>
<td>HRO</td>
<td>baseball</td>
<td>126.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>3</td>
<td>HSK</td>
<td>baseball bat</td>
<td>240.00</td>
<td>case</td>
<td>12/case</td>
</tr>
<tr>
<td>3</td>
<td>SHM</td>
<td>baseball bat</td>
<td>280.00</td>
<td>case</td>
<td>12/case</td>
</tr>
<tr>
<td>4</td>
<td>HSK</td>
<td>football</td>
<td>960.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>4</td>
<td>HRO</td>
<td>football</td>
<td>480.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>5</td>
<td>NRG</td>
<td>tennis racquet</td>
<td>28.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>5</td>
<td>SMT</td>
<td>tennis racquet</td>
<td>25.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>5</td>
<td>ANZ</td>
<td>tennis racquet</td>
<td>19.80</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>6</td>
<td>SMT</td>
<td>tennis ball</td>
<td>36.00</td>
<td>case</td>
<td>24 cans/case</td>
</tr>
<tr>
<td>6</td>
<td>ANZ</td>
<td>tennis ball</td>
<td>48.00</td>
<td>case</td>
<td>24 cans/case</td>
</tr>
<tr>
<td>7</td>
<td>HRO</td>
<td>basketball</td>
<td>600.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>8</td>
<td>ANZ</td>
<td>volleyball</td>
<td>840.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>9</td>
<td>ANZ</td>
<td>volleyball net</td>
<td>20.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>101</td>
<td>PRC</td>
<td>bicycle tires</td>
<td>88.00</td>
<td>box</td>
<td>4/box</td>
</tr>
<tr>
<td>101</td>
<td>SHM</td>
<td>bicycle tires</td>
<td>68.00</td>
<td>box</td>
<td>4/box</td>
</tr>
<tr>
<td>102</td>
<td>SHM</td>
<td>bicycle brakes</td>
<td>220.00</td>
<td>case</td>
<td>4 sets/case</td>
</tr>
<tr>
<td>102</td>
<td>PRC</td>
<td>bicycle brakes</td>
<td>480.00</td>
<td>case</td>
<td>4 sets/case</td>
</tr>
<tr>
<td>103</td>
<td>PRC</td>
<td>front derailleur</td>
<td>20.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>104</td>
<td>PRC</td>
<td>rear derailleur</td>
<td>58.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>105</td>
<td>PRC</td>
<td>bicycle wheels</td>
<td>53.00</td>
<td>pair</td>
<td>pair</td>
</tr>
<tr>
<td>105</td>
<td>SHM</td>
<td>bicycle wheels</td>
<td>80.00</td>
<td>pair</td>
<td>pair</td>
</tr>
<tr>
<td>stock_num</td>
<td>manu_code</td>
<td>description</td>
<td>unit_rice</td>
<td>unit</td>
<td>unit_descr</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>----------------------</td>
<td>-----------</td>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>106</td>
<td>PRC</td>
<td>bicycle stem</td>
<td>23.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>107</td>
<td>PRC</td>
<td>bicycle saddle</td>
<td>70.00</td>
<td>pair</td>
<td>pair</td>
</tr>
<tr>
<td>108</td>
<td>SHM</td>
<td>crankset</td>
<td>45.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>109</td>
<td>PRC</td>
<td>pedal binding</td>
<td>30.00</td>
<td>case</td>
<td>6 pairs/case</td>
</tr>
<tr>
<td>109</td>
<td>SHM</td>
<td>pedal binding</td>
<td>200.00</td>
<td>case</td>
<td>4 pairs/case</td>
</tr>
<tr>
<td>110</td>
<td>PRC</td>
<td>helmet</td>
<td>236.00</td>
<td>case</td>
<td>4/case</td>
</tr>
<tr>
<td>110</td>
<td>ANZ</td>
<td>helmet</td>
<td>244.00</td>
<td>case</td>
<td>4/case</td>
</tr>
<tr>
<td>110</td>
<td>SHM</td>
<td>helmet</td>
<td>228.00</td>
<td>case</td>
<td>4/case</td>
</tr>
<tr>
<td>110</td>
<td>HRO</td>
<td>helmet</td>
<td>260.00</td>
<td>case</td>
<td>4/case</td>
</tr>
<tr>
<td>110</td>
<td>HSK</td>
<td>helmet</td>
<td>308.00</td>
<td>case</td>
<td>4/case</td>
</tr>
<tr>
<td>111</td>
<td>SHM</td>
<td>10-spd, assmbld</td>
<td>499.99</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>112</td>
<td>SHM</td>
<td>12-spd, assmbld</td>
<td>549.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>113</td>
<td>SHM</td>
<td>18-spd, assmbld</td>
<td>685.90</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>114</td>
<td>PRC</td>
<td>bicycle gloves</td>
<td>120.00</td>
<td>case</td>
<td>10 pairs/case</td>
</tr>
<tr>
<td>201</td>
<td>NKL</td>
<td>golf shoes</td>
<td>37.50</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>201</td>
<td>ANZ</td>
<td>golf shoes</td>
<td>75.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>201</td>
<td>KAR</td>
<td>golf shoes</td>
<td>90.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>202</td>
<td>NKL</td>
<td>metal woods</td>
<td>174.00</td>
<td>case</td>
<td>2 sets/case</td>
</tr>
<tr>
<td>202</td>
<td>KAR</td>
<td>std woods</td>
<td>230.00</td>
<td>case</td>
<td>2 sets/case</td>
</tr>
<tr>
<td>203</td>
<td>NKL</td>
<td>irons/wedges</td>
<td>670.00</td>
<td>case</td>
<td>2 sets/case</td>
</tr>
<tr>
<td>204</td>
<td>KAR</td>
<td>putter</td>
<td>45.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>205</td>
<td>NKL</td>
<td>3 golf balls</td>
<td>312.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>205</td>
<td>ANZ</td>
<td>3 golf balls</td>
<td>312.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>205</td>
<td>HRO</td>
<td>3 golf balls</td>
<td>312.00</td>
<td>case</td>
<td>24/case</td>
</tr>
<tr>
<td>301</td>
<td>NKL</td>
<td>running shoes</td>
<td>97.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>301</td>
<td>HRO</td>
<td>running shoes</td>
<td>42.50</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>301</td>
<td>SHM</td>
<td>running shoes</td>
<td>102.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>301</td>
<td>PRC</td>
<td>running shoes</td>
<td>75.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>301</td>
<td>KAR</td>
<td>running shoes</td>
<td>87.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>301</td>
<td>ANZ</td>
<td>running shoes</td>
<td>95.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>302</td>
<td>HRO</td>
<td>ice pack</td>
<td>4.50</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>302</td>
<td>KAR</td>
<td>ice pack</td>
<td>5.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>303</td>
<td>PRC</td>
<td>socks</td>
<td>48.00</td>
<td>box</td>
<td>24 pairs/box</td>
</tr>
<tr>
<td>303</td>
<td>KAR</td>
<td>socks</td>
<td>36.00</td>
<td>box</td>
<td>24 pair/box</td>
</tr>
<tr>
<td>304</td>
<td>ANZ</td>
<td>watch</td>
<td>170.00</td>
<td>box</td>
<td>10/box</td>
</tr>
<tr>
<td>304</td>
<td>HRO</td>
<td>watch</td>
<td>280.00</td>
<td>box</td>
<td>10/box</td>
</tr>
<tr>
<td>305</td>
<td>HRO</td>
<td>first-aid kit</td>
<td>48.00</td>
<td>case</td>
<td>4/case</td>
</tr>
<tr>
<td>306</td>
<td>PRC</td>
<td>tandem adapter</td>
<td>160.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>306</td>
<td>SHM</td>
<td>tandem adapter</td>
<td>190.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>307</td>
<td>PRC</td>
<td>infant jogger</td>
<td>250.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>308</td>
<td>PRC</td>
<td>twin jogger</td>
<td>280.00</td>
<td>each</td>
<td>each</td>
</tr>
<tr>
<td>stock_num</td>
<td>manu_code</td>
<td>description</td>
<td>unit_rice</td>
<td>unit</td>
<td>unit_descr</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-----------------</td>
<td>-----------</td>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>309</td>
<td>HRO</td>
<td>ear drops</td>
<td>40.00</td>
<td>case</td>
<td>20/case</td>
</tr>
<tr>
<td>309</td>
<td>SHM</td>
<td>ear drops</td>
<td>40.00</td>
<td>case</td>
<td>20/case</td>
</tr>
<tr>
<td>310</td>
<td>SHM</td>
<td>kick board</td>
<td>80.00</td>
<td>case</td>
<td>10/case</td>
</tr>
<tr>
<td>310</td>
<td>ANZ</td>
<td>kick board</td>
<td>89.00</td>
<td>case</td>
<td>12/case</td>
</tr>
<tr>
<td>311</td>
<td>SHM</td>
<td>water gloves</td>
<td>48.00</td>
<td>box</td>
<td>4 pairs/box</td>
</tr>
<tr>
<td>312</td>
<td>SHM</td>
<td>racer goggles</td>
<td>96.00</td>
<td>box</td>
<td>12/box</td>
</tr>
<tr>
<td>312</td>
<td>HRO</td>
<td>racer goggles</td>
<td>72.00</td>
<td>box</td>
<td>12/box</td>
</tr>
<tr>
<td>313</td>
<td>SHM</td>
<td>swim cap</td>
<td>72.00</td>
<td>box</td>
<td>12/box</td>
</tr>
<tr>
<td>313</td>
<td>ANZ</td>
<td>swim cap</td>
<td>60.00</td>
<td>box</td>
<td>12/box</td>
</tr>
</tbody>
</table>

### Catalog Table

<table>
<thead>
<tr>
<th>catalog_num</th>
<th>stock_num</th>
<th>manu_code</th>
<th>cat_descr</th>
<th>cat_picture</th>
<th>cat_advert</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001</td>
<td>1</td>
<td>HRO</td>
<td>Brown leather. Specify first baseman’s or infield/outfield style. Specify right- or left-handed.</td>
<td>&lt;BYTE value&gt;</td>
<td>Your First Season’s Baseball Glove</td>
</tr>
<tr>
<td>10002</td>
<td>1</td>
<td>HSK</td>
<td>Babe Ruth signature glove. Black leather. Infield/outfield style. Specify right- or left-handed.</td>
<td>&lt;BYTE value&gt;</td>
<td>All-Leather, Hand-Stitched, Deep-Pockets, Sturdy Webbing that Won’t Let Go</td>
</tr>
<tr>
<td>10003</td>
<td>1</td>
<td>SMT</td>
<td>Catcher’s mitt. Brown leather. Specify right- or left-handed.</td>
<td>&lt;BYTE value&gt;</td>
<td>A Sturdy Catcher’s Mitt With the Perfect Pocket</td>
</tr>
<tr>
<td>10004</td>
<td>2</td>
<td>HRO</td>
<td>Jackie Robinson signature glove. Highest Professional quality, used by National League.</td>
<td>&lt;BYTE value&gt;</td>
<td>Highest Quality Ball Available, from the Hand-Stitching to the Robinson Signature</td>
</tr>
<tr>
<td>10005</td>
<td>3</td>
<td>HSK</td>
<td>Pro-style wood. Available in sizes: 31, 32, 33, 34, 35.</td>
<td>&lt;BYTE value&gt;</td>
<td>High-Technology Design Expands the Sweet Spot</td>
</tr>
<tr>
<td>10006</td>
<td>3</td>
<td>SHM</td>
<td>Aluminum. Blue with black tape. 31”, 20 oz or 22 oz; 32”, 21 oz or 23 oz; 33”, 22 oz or 24 oz.</td>
<td>&lt;BYTE value&gt;</td>
<td>Durable Aluminum for High School and Collegiate Athletes</td>
</tr>
<tr>
<td>10007</td>
<td>4</td>
<td>HSK</td>
<td>Norm Van Brocklin signature style.</td>
<td>&lt;BYTE value&gt;</td>
<td>Quality Pigskin with Norm Van Brocklin Signature</td>
</tr>
<tr>
<td>10008</td>
<td>4</td>
<td>HRO</td>
<td>NFL-Style pigskin.</td>
<td>&lt;BYTE value&gt;</td>
<td>Highest Quality Football for High School and Collegiate Competitions</td>
</tr>
<tr>
<td>10009</td>
<td>5</td>
<td>NRG</td>
<td>Graphite frame. Synthetic strings.</td>
<td>&lt;BYTE value&gt;</td>
<td>Wide Body Amplifies Your Natural Abilities by Providing More Power Through Aerodynamic Design</td>
</tr>
<tr>
<td>10010</td>
<td>5</td>
<td>SMT</td>
<td>Aluminum frame. Synthetic strings.</td>
<td>&lt;BYTE value&gt;</td>
<td>Mid-Sized Racquet for the Improving Player</td>
</tr>
<tr>
<td>10011</td>
<td>5</td>
<td>ANZ</td>
<td>Wood frame, cat-gut strings.</td>
<td>&lt;BYTE value&gt;</td>
<td>Antique Replica of Classic Wooden Racquet Built with Cat-Gut Strings</td>
</tr>
<tr>
<td>10012</td>
<td>6</td>
<td>SMT</td>
<td>Soft yellow color for easy visibility in sunlight or artificial light.</td>
<td>&lt;BYTE value&gt;</td>
<td>High-Visibility Tennis, Day or Night</td>
</tr>
<tr>
<td>catalog_num</td>
<td>stock_num</td>
<td>manu_code</td>
<td>cat_descr</td>
<td>cat_picture</td>
<td>cat_advert</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------</td>
<td>-------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>10013</td>
<td>6</td>
<td>ANZ</td>
<td>Pro-core. Available in neon yellow, green, and pink.</td>
<td>&lt;BYTE value&gt;</td>
<td>Durable Construction Coupled with the Brightest Colors Available</td>
</tr>
<tr>
<td>10014</td>
<td>7</td>
<td>HRO</td>
<td>Indoor. Classic NBA style. Brown leather.</td>
<td>&lt;BYTE value&gt;</td>
<td>Long-Life Basketballs for Indoor Gymnasiums</td>
</tr>
<tr>
<td>10015</td>
<td>8</td>
<td>ANZ</td>
<td>Indoor. Finest leather. Professional quality.</td>
<td>&lt;BYTE value&gt;</td>
<td>Professional Volleyballs for Indoor Competitions</td>
</tr>
<tr>
<td>10016</td>
<td>9</td>
<td>ANZ</td>
<td>Steel eyelets. Nylon cording. Double-stitched. Sanctioned by the National Athletic Congress.</td>
<td>&lt;BYTE value&gt;</td>
<td>Sanctioned Volleyball Netting for Indoor Professional and Collegiate Competition</td>
</tr>
<tr>
<td>10017</td>
<td>101</td>
<td>PRC</td>
<td>Reinforced, hand-finished tubular. Polyurethane belted. Effective against punctures. Mixed tread for super wear and road grip.</td>
<td>&lt;BYTE value&gt;</td>
<td>Ultimate in Puncture Protection, Tires Designed for In-City Riding</td>
</tr>
<tr>
<td>10018</td>
<td>101</td>
<td>SHM</td>
<td>Durable nylon casing with butyl tube for superior air retention. Center-ribbed tread with herringbone side. Coated sidewalls resist abrasion.</td>
<td>&lt;BYTE value&gt;</td>
<td>The Perfect Tire for Club Rides or Training</td>
</tr>
<tr>
<td>10019</td>
<td>102</td>
<td>SHM</td>
<td>Thrust bearing and coated pivot washer/ spring sleeve for smooth action. Slotted levers with soft gum hoods. Two-tone paint treatment. Set includes calipers, levers, and cables.</td>
<td>&lt;BYTE value&gt;</td>
<td>Thrust-Bearing and Spring-Sleeve Brake Set Guarantees Smooth Action</td>
</tr>
<tr>
<td>10020</td>
<td>102</td>
<td>PRC</td>
<td>Computer-aided design with low-profile pads. Cold-forged alloy calipers and beefy caliper bushing. Aero levers. Set includes calipers, levers, and cables.</td>
<td>&lt;BYTE value&gt;</td>
<td>Computer Design Delivers Rigid Yet Vibration-Free Brakes</td>
</tr>
<tr>
<td>10021</td>
<td>103</td>
<td>PRC</td>
<td>Compact leading-action design enhances shifting. Deep cage for super-small granny gears. Extra strong construction to resist off-road abuse.</td>
<td>&lt;BYTE value&gt;</td>
<td>Climb Any Mountain: ProCycle’s Front Derailleur Adds Finesse to Your ATB</td>
</tr>
<tr>
<td>10022</td>
<td>104</td>
<td>PRC</td>
<td>Floating trapezoid geometry with extra thick parallelogram arms. 100-tooth capacity. Optimum alignment with any freewheel.</td>
<td>&lt;BYTE value&gt;</td>
<td>Computer-Aided Design Engineers 100-Tooth Capacity Into ProCycle’s Rear Derailleur</td>
</tr>
<tr>
<td>10023</td>
<td>105</td>
<td>PRC</td>
<td>Front wheels laced with 15g spokes in a 3-cross pattern. Rear wheels laced with 14g spikes in a 3-cross pattern.</td>
<td>&lt;BYTE value&gt;</td>
<td>Durable Training Wheels That Hold True Under Toughest Conditions</td>
</tr>
<tr>
<td>10024</td>
<td>105</td>
<td>SHM</td>
<td>Polished alloy. Sealed-bearing, quick-release hubs. Double-butted. Front wheels are laced 15g/2-cross. Rear wheels are laced 15g/3-cross.</td>
<td>&lt;BYTE value&gt;</td>
<td>Extra Lightweight Wheels for Training or High-Performance Touring</td>
</tr>
<tr>
<td>10025</td>
<td>106</td>
<td>PRC</td>
<td>Hard anodized alloy with pearl finish. 6mm hex bolt hardware. Available in lengths of 90-140mm in 10mm increments.</td>
<td>&lt;BYTE value&gt;</td>
<td>ProCycle Stem with Pearl Finish</td>
</tr>
<tr>
<td>catalog_num</td>
<td>stock_num</td>
<td>manu_code</td>
<td>cat_descr</td>
<td>cat_picture</td>
<td>cat_advert</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>10026</td>
<td>107</td>
<td>PRC</td>
<td>Available in three styles: Men’s racing; Men’s touring; and Women’s. Anatomical gel construction with lycra cover. Black or black/hot pink.</td>
<td>&lt;BYTE value&gt;</td>
<td>The Ultimate In Riding Comfort, Lightweight With Anatomical Support</td>
</tr>
<tr>
<td>10027</td>
<td>108</td>
<td>SHM</td>
<td>Double or triple crankset with choice of chainrings. For double crankset, chainrings from 38-54 teeth. For triple crankset, chainrings from 24-48 teeth.</td>
<td>&lt;BYTE value&gt;</td>
<td>Customize Your Mountain Bike With Extra-Durable Crankset</td>
</tr>
<tr>
<td>10028</td>
<td>109</td>
<td>PRC</td>
<td>Steel toe clips with nylon strap. Extra wide at buckle to reduce pressure.</td>
<td>&lt;BYTE value&gt;</td>
<td>Classic Toeclip Improved to Prevent Soreness at Clip Buckle</td>
</tr>
<tr>
<td>10029</td>
<td>109</td>
<td>SHM</td>
<td>Ingenious new design combines button on sole of shoe with slot on a pedal plate to give riders new options in riding efficiency. Choose full or partial locking. Four plates mean both top and bottom of pedals are slotted—no fishing around when you want to engage full power. Fast unlocking ensures safety when maneuverability is paramount.</td>
<td>&lt;BYTE value&gt;</td>
<td>Ingenious Pedal/Clip Design Delivers Maximum Power and Fast Unlocking</td>
</tr>
<tr>
<td>10030</td>
<td>110</td>
<td>PRC</td>
<td>Super-lightweight. Meets both ANSI and Snell standards for impact protection. 7.5 oz. Quick-release shadow buckle.</td>
<td>&lt;BYTE value&gt;</td>
<td>Feather-Light, Quick-Release, Maximum Protection Helmet</td>
</tr>
<tr>
<td>10031</td>
<td>110</td>
<td>ANZ</td>
<td>No buckle so no plastic touches your chin. Meets both ANSI and Snell standards for impact protection. 7.5 oz. Lycra cover.</td>
<td>&lt;BYTE value&gt;</td>
<td>Minimum Chin Contact, Feather-Light, Maximum Protection Helmet</td>
</tr>
<tr>
<td>10032</td>
<td>110</td>
<td>SHM</td>
<td>Dense outer layer combines with softer inner layer to eliminate the mesh cover, no snagging on brush. Meets both ANSI and Snell standards for impact protection. 8.0 oz.</td>
<td>&lt;BYTE value&gt;</td>
<td>Mountain Bike Helmet: Smooth Cover Eliminates the Worry of Brush Snags But Delivers Maximum Protection</td>
</tr>
<tr>
<td>10033</td>
<td>110</td>
<td>HRO</td>
<td>Newest ultralight helmet uses plastic shell. Largest ventilation channels of any helmet on the market. 8.5 oz.</td>
<td>&lt;BYTE value&gt;</td>
<td>Lightweight Plastic with Vents Assures Cool Comfort Without Sacrificing Protection</td>
</tr>
<tr>
<td>10034</td>
<td>110</td>
<td>HSK</td>
<td>Aerodynamic (teardrop) helmet covered with anti-drag fabric. Credited with shaving 2 seconds/mile from winner’s time in Tour de France time-trial. 7.5 oz.</td>
<td>&lt;BYTE value&gt;</td>
<td>Teardrop Design Used by Yellow Jerseys, You Can Time the Difference</td>
</tr>
<tr>
<td>10035</td>
<td>111</td>
<td>SHM</td>
<td>Light-action shifting 10 speed. Designed for the city commuter with shock-absorbing front fork and drilled eyelets for carry-all racks or bicycle trailers. Internal wiring for generator lights. 33 lbs.</td>
<td>&lt;BYTE value&gt;</td>
<td>Fully Equipped Bicycle Designed for the Serious Commuter Who Mixes Business With Pleasure</td>
</tr>
<tr>
<td>catalog_num</td>
<td>stock_num</td>
<td>manu_code</td>
<td>cat_descr</td>
<td>cat_picture</td>
<td>cat_advert</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-----------</td>
<td>---------------------------------------------------------------------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>10036</td>
<td>112</td>
<td>SHM</td>
<td>Created for the beginner enthusiast. Ideal for club rides and light touring. Sophisticated triple-butted frame construction. Precise index shifting. 28 lbs.</td>
<td>&lt;BYTE value&gt;</td>
<td>We Selected the Ideal Combination of Touring Bike Equipment, then Turned It Into This Package Deal: High-Performance on the Roads, Maximum Pleasure Everywhere</td>
</tr>
<tr>
<td>10038</td>
<td>114</td>
<td>PRC</td>
<td>Padded leather palm and stretch mesh merged with Terry back; Available in tan, black, and cream. Sizes S, M, L, XL.</td>
<td>&lt;BYTE value&gt;</td>
<td>Riding Gloves for Comfort and Protection</td>
</tr>
<tr>
<td>10039</td>
<td>201</td>
<td>NKL</td>
<td>Designed for comfort and stability. Available in white &amp; blue or white &amp; brown. Specify size.</td>
<td>&lt;BYTE value&gt;</td>
<td>Full-Comfort, Long-Wearing Golf Shoes for Men and Women</td>
</tr>
<tr>
<td>10040</td>
<td>201</td>
<td>ANZ</td>
<td>Guaranteed waterproof. Full leather upper. Available in white, bone, brown, green, and blue. Specify size.</td>
<td>&lt;BYTE value&gt;</td>
<td>Waterproof Protection Ensures Maximum Comfort and Durability In All Climates</td>
</tr>
<tr>
<td>10041</td>
<td>201</td>
<td>KAR</td>
<td>Leather and leather mesh for maximum ventilation. Waterproof lining to keep feet dry. Available in white and gray or white and ivory. Specify size.</td>
<td>&lt;BYTE value&gt;</td>
<td>Karsten’s Top Quality Shoe Combines Leather and Leather Mesh</td>
</tr>
<tr>
<td>10042</td>
<td>202</td>
<td>NKL</td>
<td>Complete starter set utilizes gold shafts. Balanced for power.</td>
<td>&lt;BYTE value&gt;</td>
<td>Starter Set of Woods, Ideal for High School and Collegiate Classes</td>
</tr>
<tr>
<td>10043</td>
<td>202</td>
<td>KAR</td>
<td>Full set of woods designed for precision control and power performance.</td>
<td>&lt;BYTE value&gt;</td>
<td>High-Quality Woods Appropriate for High School Competitions or Serious Amateurs</td>
</tr>
<tr>
<td>10044</td>
<td>203</td>
<td>NKL</td>
<td>Set of eight irons includes 3 through 9 irons and pitching wedge. Originally priced at $489.00.</td>
<td>&lt;BYTE value&gt;</td>
<td>Set of Irons Available From Factory at Tremendous Savings: Discontinued Line</td>
</tr>
<tr>
<td>10045</td>
<td>204</td>
<td>KAR</td>
<td>Ideally balanced for optimum control. Nylon-covered shaft.</td>
<td>&lt;BYTE value&gt;</td>
<td>High-Quality Beginning Set of Irons Appropriate for High School Competitions</td>
</tr>
<tr>
<td>10046</td>
<td>205</td>
<td>NKL</td>
<td>Fluorescent yellow.</td>
<td>&lt;BYTE value&gt;</td>
<td>Long Drive Golf Balls: Fluorescent Yellow</td>
</tr>
<tr>
<td>10047</td>
<td>205</td>
<td>ANZ</td>
<td>White only.</td>
<td>&lt;BYTE value&gt;</td>
<td>Long Drive Golf Balls: White</td>
</tr>
<tr>
<td>10048</td>
<td>205</td>
<td>HRO</td>
<td>Combination fluorescent yellow and standard white.</td>
<td>&lt;BYTE value&gt;</td>
<td>HiFlier Golf Balls: Case Includes Fluorescent Yellow and Standard White</td>
</tr>
<tr>
<td>catalog_num</td>
<td>stock_num</td>
<td>manu_code</td>
<td>cat_descr</td>
<td>cat_picture</td>
<td>cat_advert</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>-----------</td>
<td>--------------------------------------------------------------------------</td>
<td>-------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>10049</td>
<td>301</td>
<td>NKL</td>
<td>Super shock-absorbing gel pads disperse vertical energy into a horizontal plane for extraordinary cushioned comfort. Great motion control. Men’s only. Specify size.</td>
<td>&lt;BYTE value&gt;</td>
<td>Maximum Protection For High-Mileage Runners</td>
</tr>
<tr>
<td>10050</td>
<td>301</td>
<td>HRO</td>
<td>Engineered for serious training with exceptional stability. Fabulous shock absorption. Great durability. Specify men’s/women’s size.</td>
<td>&lt;BYTE value&gt;</td>
<td>Pronators and Supinators Take Heart: A Serious Training Shoe For Runners Who Need Motion Control</td>
</tr>
<tr>
<td>10051</td>
<td>301</td>
<td>SHM</td>
<td>For runners who log heavy miles and need a durable, supportive, stable platform. Mesh/synthetic upper gives excellent moisture dissipation. Stability system uses rear antipronation platform and forefoot control plate for extended protection during high-intensity training. Specify men’s/women’s size.</td>
<td>&lt;BYTE value&gt;</td>
<td>The Training Shoe Engineered for Marathoners and Ultra-Distance Runners</td>
</tr>
<tr>
<td>10052</td>
<td>301</td>
<td>PRC</td>
<td>Supportive, stable racing flat. Plenty of forefoot cushioning with added motion control. Women’s only. D widths available. Specify size.</td>
<td>&lt;BYTE value&gt;</td>
<td>A Woman’s Racing Flat That Combines Extra Forefoot Protection With a Slender Heel</td>
</tr>
<tr>
<td>10053</td>
<td>301</td>
<td>KAR</td>
<td>Anatomical last holds your foot firmly in place. Feather-weight cushioning delivers the responsiveness of a racing flat. Specify men’s/women’s size.</td>
<td>&lt;BYTE value&gt;</td>
<td>Durable Training Flat That Can Carry You Through Marathon Miles</td>
</tr>
<tr>
<td>10054</td>
<td>301</td>
<td>ANZ</td>
<td>Cantilever sole provides shock absorption and energy rebound. Positive traction shoe with ample toe box. Ideal for runners who need a wide shoe. Available in men’s and women’s. Specify size.</td>
<td>&lt;BYTE value&gt;</td>
<td>Motion Control, Protection, and Extra Toebox Room</td>
</tr>
<tr>
<td>10055</td>
<td>302</td>
<td>KAR</td>
<td>Reusable ice pack with velcro strap. For general use. Velcro strap allows easy application to arms or legs.</td>
<td>&lt;BYTE value&gt;</td>
<td>Finally, an Ice Pack for Achilles Injuries and Shin Splints That You Can Take to the Office</td>
</tr>
<tr>
<td>10056</td>
<td>303</td>
<td>PRC</td>
<td>Neon nylon. Perfect for running or aerobics. Indicate color: Fluorescent pink, yellow, green, and orange.</td>
<td>&lt;BYTE value&gt;</td>
<td>Knock Their Socks Off With YOUR Socks</td>
</tr>
<tr>
<td>10057</td>
<td>303</td>
<td>KAR</td>
<td>100% nylon blend for optimal wicking and comfort. We’ve taken out the cotton to eliminate the risk of blisters and reduce the opportunity for infection. Specify men’s or women’s.</td>
<td>&lt;BYTE value&gt;</td>
<td>100% Nylon Blend Socks - No Cotton</td>
</tr>
<tr>
<td>10058</td>
<td>304</td>
<td>ANZ</td>
<td>Provides time, date, dual display of lap/cumulative splits, 4-lap memory, 10 hr count-down timer, event timer, alarm, hour chime, waterproof to 50m, velcro band.</td>
<td>&lt;BYTE value&gt;</td>
<td>Athletic Watch w/4-Lap Memory</td>
</tr>
</tbody>
</table>

Appendix A. The stores_demo Database  A-21
<table>
<thead>
<tr>
<th>catalog_num</th>
<th>stock_num</th>
<th>manu_code</th>
<th>catdescr</th>
<th>cat_picture</th>
<th>cat_advert</th>
</tr>
</thead>
<tbody>
<tr>
<td>10059</td>
<td>304</td>
<td>HRO</td>
<td>Split timer, waterproof to 50m. Indicate color: Hot pink, mint, green, space black.</td>
<td>BYTE value</td>
<td>Waterproof Triathlete Watch In Competition Colors</td>
</tr>
<tr>
<td>10060</td>
<td>305</td>
<td>HRO</td>
<td>Contains ace bandage, anti-bacterial cream, alcohol cleansing pads, adhesive bandages of assorted sizes, and instant-cold pack.</td>
<td>BYTE value</td>
<td>Comprehensive First-Aid Kit Essential for Team Practices, Team Traveling</td>
</tr>
<tr>
<td>10061</td>
<td>306</td>
<td>PRC</td>
<td>Converts a standard tandem bike into an adult/child bike. User-tested assembly instructions</td>
<td>BYTE value</td>
<td>Enjoy Bicycling With Your Child on a Tandem; Make Your Family Outing Safer</td>
</tr>
<tr>
<td>10062</td>
<td>306</td>
<td>SHM</td>
<td>Converts a standard tandem bike into an adult/child bike. Lightweight model.</td>
<td>BYTE value</td>
<td>Consider a Touring Vacation for the Entire Family: A Lightweight, Touring Tandem for Parent and Child</td>
</tr>
<tr>
<td>10063</td>
<td>307</td>
<td>PRC</td>
<td>Allows mom or dad to take the baby out too. Fits children up to 21 pounds. Navy blue with black trim.</td>
<td>BYTE value</td>
<td>Infant Jogger Keeps A Running Family Together</td>
</tr>
<tr>
<td>10064</td>
<td>308</td>
<td>PRC</td>
<td>Allows mom or dad to take both children! Rated for children up to 18 pounds.</td>
<td>BYTE value</td>
<td>As Your Family Grows, Infant Jogger Grows With You</td>
</tr>
<tr>
<td>10065</td>
<td>309</td>
<td>HRO</td>
<td>Prevents swimmer’s ear.</td>
<td>BYTE value</td>
<td>Swimmers Can Prevent Ear Infection All Season Long</td>
</tr>
<tr>
<td>10066</td>
<td>309</td>
<td>SHM</td>
<td>Extra-gentle formula. Can be used every day for prevention or treatment of swimmer’s ear.</td>
<td>BYTE value</td>
<td>Swimmer’s Ear Drops Specially Formulated for Children</td>
</tr>
<tr>
<td>10067</td>
<td>310</td>
<td>SHM</td>
<td>Blue heavy-duty foam board with Shimara or team logo.</td>
<td>BYTE value</td>
<td>Exceptionally Durable, Compact Kickboard for Team Practice</td>
</tr>
<tr>
<td>10068</td>
<td>310</td>
<td>ANZ</td>
<td>White. Standard size.</td>
<td>BYTE value</td>
<td>High-Quality Kickboard</td>
</tr>
<tr>
<td>10069</td>
<td>311</td>
<td>SHM</td>
<td>Swim gloves. Webbing between fingers promotes strengthening of arms. Cannot be used in competition.</td>
<td>BYTE value</td>
<td>Hot Training Tool - Webbed Swim Gloves Build Arm Strength and Endurance</td>
</tr>
<tr>
<td>10070</td>
<td>312</td>
<td>SHM</td>
<td>Hydrodynamic egg-shaped lens. Ground-in anti-fog elements; Available in blue or smoke.</td>
<td>BYTE value</td>
<td>Anti-Fog Swimmer’s Goggles: Quantity Discount</td>
</tr>
<tr>
<td>10071</td>
<td>312</td>
<td>HRO</td>
<td>Durable competition-style goggles. Available in blue, grey, or white.</td>
<td>BYTE value</td>
<td>Swim Goggles: Traditional Rounded Lens For Greater Comfort</td>
</tr>
<tr>
<td>10072</td>
<td>313</td>
<td>SHM</td>
<td>Silicone swim cap. One size. Available in white, silver, or navy. Team Logo Imprinting Available.</td>
<td>BYTE value</td>
<td>Team Logo Silicone Swim Cap</td>
</tr>
<tr>
<td>10073</td>
<td>314</td>
<td>ANZ</td>
<td>Silicone swim cap. Squared-off top. One size. White</td>
<td>BYTE value</td>
<td>Durable Squared-off Silicone Swim Cap</td>
</tr>
<tr>
<td>10074</td>
<td>315</td>
<td>HRO</td>
<td>Re-usable ice pack. Store in the freezer for instant first-aid. Extra capacity to accommodate water and ice.</td>
<td>BYTE value</td>
<td>Water Compartment Combines With Ice to Provide Optimal Orthopedic Treatment</td>
</tr>
</tbody>
</table>
### cust_calls Table

<table>
<thead>
<tr>
<th>customer_num</th>
<th>call_dtime</th>
<th>user_id</th>
<th>call_code</th>
<th>call_descr</th>
<th>res_dtime</th>
<th>res_descr</th>
</tr>
</thead>
<tbody>
<tr>
<td>106</td>
<td>1998-06-12 8:20</td>
<td>maryj</td>
<td>D</td>
<td>Order was received, but two of the cans of ANZ tennis balls within the case were empty.</td>
<td>1998-06-12 8:25</td>
<td>Authorized credit for two cans to customer, issued apology. Called ANZ buyer to report the QA problem.</td>
</tr>
<tr>
<td>110</td>
<td>1998-07-07 10:24</td>
<td>richc</td>
<td>L</td>
<td>Order placed one month ago (6/7) not received.</td>
<td>1998-07-07 10:30</td>
<td>Checked with shipping (Ed Smith). Order sent yesterday - we were waiting for goods from ANZ. Next time will call with delay if necessary.</td>
</tr>
<tr>
<td>119</td>
<td>1998-07-01 15:00</td>
<td>richc</td>
<td>B</td>
<td>Bill does not reflect credit from previous order.</td>
<td>1998-07-02 8:21</td>
<td>Spoke with Jane Akant in Finance. She found the error and is sending new bill to customer.</td>
</tr>
<tr>
<td>121</td>
<td>1998-07-10 14:05</td>
<td>maryj</td>
<td>O</td>
<td>Customer likes our merchandise. Requests that we stock more types of infant joggers. Will call back to place order.</td>
<td>1998-07-10 14:06</td>
<td>Sent note to marketing group of interest in infant joggers.</td>
</tr>
<tr>
<td>127</td>
<td>1998-07-31 14:30</td>
<td>maryj</td>
<td>I</td>
<td>Received Hero watches (item # 304) instead of ANZ watches.</td>
<td></td>
<td>Sent memo to shipping to send ANZ item 304 to customer and pickup HRO watches. Should be done tomorrow, 8/1.</td>
</tr>
<tr>
<td>116</td>
<td>1997-11-28 13:34</td>
<td>manyn</td>
<td>I</td>
<td>Received plain white swim caps (313 ANZ) instead of navy with team logo (313 SHM).</td>
<td>1997-11-28 16:47</td>
<td>Shipping found correct case in warehouse and express mailed it in time for swim meet.</td>
</tr>
<tr>
<td>116</td>
<td>1997-12-21 11:24</td>
<td>manyn</td>
<td>I</td>
<td>Second complaint from this customer! Received two cases right-handed outfielder gloves (1 HRO) instead of one case lefties.</td>
<td>1997-12-27 08:19</td>
<td>Memo to shipping (Ava Brown) to send case of left-handed gloves, pick up wrong case; memo to billing requesting 5% discount to placate customer due to second offense and lateness of resolution because of holiday.</td>
</tr>
</tbody>
</table>
### manufacture Table

<table>
<thead>
<tr>
<th>manu_code</th>
<th>manu_name</th>
<th>lead_time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANZ</td>
<td>Anza</td>
<td>5</td>
</tr>
<tr>
<td>HSK</td>
<td>Husky</td>
<td>5</td>
</tr>
<tr>
<td>HRO</td>
<td>Hero</td>
<td>4</td>
</tr>
<tr>
<td>NRG</td>
<td>Norge</td>
<td>7</td>
</tr>
<tr>
<td>SMT</td>
<td>Smith</td>
<td>3</td>
</tr>
<tr>
<td>SHM</td>
<td>Shimara</td>
<td>30</td>
</tr>
<tr>
<td>KAR</td>
<td>Karsten</td>
<td>21</td>
</tr>
<tr>
<td>NKL</td>
<td>Nikulus</td>
<td>8</td>
</tr>
<tr>
<td>PRC</td>
<td>ProCycle</td>
<td>9</td>
</tr>
</tbody>
</table>

### state Table

<table>
<thead>
<tr>
<th>code</th>
<th>sname</th>
<th>code</th>
<th>sname</th>
</tr>
</thead>
<tbody>
<tr>
<td>AK</td>
<td>Alaska</td>
<td>MT</td>
<td>Montana</td>
</tr>
<tr>
<td>AL</td>
<td>Alabama</td>
<td>NE</td>
<td>Nebraska</td>
</tr>
<tr>
<td>AR</td>
<td>Arkansas</td>
<td>NC</td>
<td>North Carolina</td>
</tr>
<tr>
<td>AZ</td>
<td>Arizona</td>
<td>ND</td>
<td>North Dakota</td>
</tr>
<tr>
<td>CA</td>
<td>California</td>
<td>NH</td>
<td>New Hampshire</td>
</tr>
<tr>
<td>CT</td>
<td>Connecticut</td>
<td>NJ</td>
<td>New Jersey</td>
</tr>
<tr>
<td>CO</td>
<td>Colorado</td>
<td>NM</td>
<td>New Mexico</td>
</tr>
<tr>
<td>DC</td>
<td>Washington, D.C.</td>
<td>NV</td>
<td>Nevada</td>
</tr>
<tr>
<td>DE</td>
<td>Delaware</td>
<td>NY</td>
<td>New York</td>
</tr>
<tr>
<td>FL</td>
<td>Florida</td>
<td>OH</td>
<td>Ohio</td>
</tr>
<tr>
<td>GA</td>
<td>Georgia</td>
<td>OK</td>
<td>Oklahoma</td>
</tr>
<tr>
<td>HI</td>
<td>Hawaii</td>
<td>OR</td>
<td>Oregon</td>
</tr>
<tr>
<td>IA</td>
<td>Iowa</td>
<td>PA</td>
<td>Pennsylvania</td>
</tr>
<tr>
<td>ID</td>
<td>Idaho</td>
<td>PR</td>
<td>Puerto Rico</td>
</tr>
<tr>
<td>IL</td>
<td>Illinois</td>
<td>RI</td>
<td>Rhode Island</td>
</tr>
<tr>
<td>IN</td>
<td>Indiana</td>
<td>SC</td>
<td>South Carolina</td>
</tr>
<tr>
<td>KY</td>
<td>Kentucky</td>
<td>TN</td>
<td>Tennessee</td>
</tr>
<tr>
<td>LA</td>
<td>Louisiana</td>
<td>TX</td>
<td>Texas</td>
</tr>
<tr>
<td>MA</td>
<td>Massachusetts</td>
<td>UT</td>
<td>Utah</td>
</tr>
<tr>
<td>MD</td>
<td>Maryland</td>
<td>VA</td>
<td>Virginia</td>
</tr>
<tr>
<td>ME</td>
<td>Maine</td>
<td>VT</td>
<td>Vermont</td>
</tr>
<tr>
<td>MI</td>
<td>Michigan</td>
<td>WA</td>
<td>Washington</td>
</tr>
<tr>
<td>MN</td>
<td>Minnesota</td>
<td>WI</td>
<td>Wisconsin</td>
</tr>
<tr>
<td>MO</td>
<td>Missouri</td>
<td>WV</td>
<td>West Virginia</td>
</tr>
<tr>
<td>MS</td>
<td>Mississippi</td>
<td>WY</td>
<td>Wyoming</td>
</tr>
</tbody>
</table>
Appendix B. The sales_demo and superstores_demo Databases

In addition to the stores_demo database that is described in detail in Appendix A, IBM Informix products include the following demonstration databases:

<table>
<thead>
<tr>
<th>Extended Parallel Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>- The sales_demo database illustrates a dimensional schema for data-warehousing applications.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamic Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>- The superstores_demo database illustrates an object-relational schema.</td>
</tr>
</tbody>
</table>

This appendix discusses the structures of these two demonstration databases.

For information on how to create and populate the demonstration databases, including relevant SQL files, see the IBM Informix DB–Access User’s Guide. For conceptual information about demonstration databases, see the IBM Informix Database Design and Implementation Guide.

The sales_demo Database (XPS)

Your database server product contains SQL scripts for the sales_demo dimensional database. The sales_demo database provides an example of a simple data-warehousing environment and works in conjunction with the stores_demo database. The scripts for the sales_demo database create new tables and add extra rows to the items and orders tables of stores_demo.

To create the sales_demo database, you must first create the stores_demo database with the logging option. Once you create the stores_demo database, you can execute the scripts that create and load the sales_demo database from DB–Access. The files are named createdw.sql and loaddw.sql.

Dimensional Model of the sales_demo Database

[Figure B-1 on page B-2] gives an overview of the tables in the sales_demo database.
For information on how to create and populate the sales_demo database, see the IBM Informix DB–Access User’s Guide. For information on how to design and implement dimensional databases, see the IBM Informix Database Design and Implementation Guide. For information on the stores_demo database, see Appendix A.

The next section describes the schema of these five tables, and identifies the column in the fact table that logically associates each dimension table to the fact table.

**Structure of the sales_demo Tables**

The sales_demo database includes the following tables:

- customer
- geography
- product
- sales
- time

The tables are listed alphabetically, not in the order in which they are created. The customer, geography, product, and time tables are the dimensions for the sales fact table.

The sales_demo database is not an ANSI-compliant database.

The following sections describe the column names, data types, and column descriptions for each table. A SERIAL field serves as the primary key for the district code column of the geography table. The primary and foreign key relationships that exist between the fact (sales) table and its dimension tables are not defined, however, because data-loading performance improves dramatically when the database server does not enforce constraint checking.
The customer Table
The customer table contains information about sales customers. Table B-1 shows the columns of the customer table.

Table B-1. The customer Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_code</td>
<td>INTEGER</td>
<td>Customer code</td>
</tr>
<tr>
<td>customer_name</td>
<td>CHAR(31)</td>
<td>Customer name</td>
</tr>
<tr>
<td>company_name</td>
<td>CHAR(20)</td>
<td>Company name</td>
</tr>
</tbody>
</table>

The geography Table
The geography table contains information about the sales district and region. Table B-2 shows the columns of the geography table.

Table B-2. The geography Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>district_code</td>
<td>SERIAL</td>
<td>District code</td>
</tr>
<tr>
<td>district_name</td>
<td>CHAR(15)</td>
<td>District name</td>
</tr>
<tr>
<td>state_code</td>
<td>CHAR(2)</td>
<td>State code</td>
</tr>
<tr>
<td>state_name</td>
<td>CHAR(18)</td>
<td>State name</td>
</tr>
<tr>
<td>region</td>
<td>SMALLINT</td>
<td>Region name</td>
</tr>
</tbody>
</table>

The product Table
The product table contains information about the products sold through the data warehouse. Table B-3 shows the columns of the product table.

Table B-3. The product Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>product_code</td>
<td>INTEGER</td>
<td>Product code</td>
</tr>
<tr>
<td>product_name</td>
<td>CHAR(31)</td>
<td>Product name</td>
</tr>
<tr>
<td>vendor_code</td>
<td>CHAR(3)</td>
<td>Vendor code</td>
</tr>
<tr>
<td>vendor_name</td>
<td>CHAR(15)</td>
<td>Vendor name</td>
</tr>
<tr>
<td>product_line_code</td>
<td>SMALLINT</td>
<td>Product line code</td>
</tr>
<tr>
<td>product_line_name</td>
<td>CHAR(15)</td>
<td>Name of product line</td>
</tr>
</tbody>
</table>

The sales Table
The sales fact table contains information about product sales and has a pointer to each dimension table. For example, the customer_code column references the customer table, the district_code column references the geography table, and so on. The sales table also contains the measures for the units sold, revenue, cost, and net profit. Table B-4 on page B-4 shows the columns of the sales table.
### Table B-4. The sales Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_code</td>
<td>INTEGER</td>
<td>Customer code (references customer)</td>
</tr>
<tr>
<td>district_code</td>
<td>SMALLINT</td>
<td>District code (references geography)</td>
</tr>
<tr>
<td>time_code</td>
<td>INTEGER</td>
<td>Time code (references time)</td>
</tr>
<tr>
<td>product_code</td>
<td>INTEGER</td>
<td>Product code (references product)</td>
</tr>
<tr>
<td>units_sold</td>
<td>SMALLINT</td>
<td>Number of units sold</td>
</tr>
<tr>
<td>revenue</td>
<td>MONEY(8,2)</td>
<td>Amount of sales revenue</td>
</tr>
<tr>
<td>cost</td>
<td>MONEY(8,2)</td>
<td>Cost of sale</td>
</tr>
<tr>
<td>net_profit</td>
<td>MONEY(8,2)</td>
<td>Net profit of sale</td>
</tr>
</tbody>
</table>

### The time Table

The time table contains time information about the sale. Table B-5 shows the columns of the time table.

### Table B-5. The time Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time_code</td>
<td>INTEGER</td>
<td>Time code</td>
</tr>
<tr>
<td>order_date</td>
<td>DATE</td>
<td>Order date</td>
</tr>
<tr>
<td>month_code</td>
<td>SMALLINT</td>
<td>Month code</td>
</tr>
<tr>
<td>month_name</td>
<td>CHAR(10)</td>
<td>Name of month</td>
</tr>
<tr>
<td>quarter_code</td>
<td>SMALLINT</td>
<td>Quarter code</td>
</tr>
<tr>
<td>quarter_name</td>
<td>CHAR(10)</td>
<td>Name of quarter</td>
</tr>
<tr>
<td>year</td>
<td>INTEGER</td>
<td>Year</td>
</tr>
</tbody>
</table>

### The superstores_demo Database (IDS)

SQL files and user-defined routines (UDRs) that are provided with DB–Access let you derive the superstores_demo object-relational database.

The superstores_demo database uses the default locale and is not ANSI compliant.

This section provides the following superstores_demo information:
- The structure of all the tables in the superstores_demo database
- A list and definition of the extended data types that superstores_demo uses
- A map of table hierarchies
- The primary-foreign key relationships among the columns in the database tables

For information on how to create and populate the superstores_demo database, see the IBM Informix DB–Access User’s Guide. For information on how to work with object-relational databases, see the IBM Informix Database Design and Implementation Guide. For information on the stores_demo database on which superstores_demo is based, see Appendix A.
Structure of the superstores_demo Tables

The superstores_demo database includes the following tables. Although many tables have the same name as stores_demo tables, they are different. The tables are listed alphabetically, not in the order in which they are created.

- call_type
- catalog
- cust_calls
- customer
  - retail_customer (new)
  - whlsale_customer (new)
- items
- location (new)
  - location_non_us (new)
  - location_us (new)
- manufact
- orders
- region (new)
- sales_rep (new)
- state
- stock
- stock_discount (new)
- units (new)

This section lists the names, data types, and descriptions of the columns for each table in the superstores_demo database. The unique identifying value for each table (primary key) is shaded. Columns that represent extended data types are discussed in "User-Defined Routines and Extended Data Types" on page B-13. Primary-foreign key relationships between the tables are outlined in "Referential Relationships" on page B-15.

The call_type Table

The call codes associated with customer calls are stored in the call_type table. Table B-6 shows the columns of the call_type table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>call_code</td>
<td>CHAR(1)</td>
<td>Call code</td>
</tr>
<tr>
<td>codel_descr</td>
<td>CHAR (30)</td>
<td>Description of call code</td>
</tr>
</tbody>
</table>

The catalog Table

The catalog table describes each item in stock. Retail stores use this table when placing orders with the distributor. Table B-7 on page B-6 shows the columns of the catalog table.
Table B-7. The catalog Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>catalog_num</td>
<td>SERIAL(1001)</td>
<td>System-generated catalog number</td>
</tr>
<tr>
<td>stock_num</td>
<td>SMALLINT</td>
<td>Distributor stock number (foreign key to stock table)</td>
</tr>
<tr>
<td>manu_code</td>
<td>CHAR(3)</td>
<td>Manufacturer code (foreign key to stock table)</td>
</tr>
<tr>
<td>unit</td>
<td>CHAR(4)</td>
<td>Unit by which item is ordered (foreign key to stock table)</td>
</tr>
<tr>
<td>advert</td>
<td>ROW (picture BLOB, caption LVARCHAR)</td>
<td>Picture of item and caption</td>
</tr>
<tr>
<td>advert_descr</td>
<td>CLOB</td>
<td>Tag line underneath picture</td>
</tr>
</tbody>
</table>

The cust_calls Table

All customer calls for information on orders, shipments, or complaints are logged.
The cust_calls table contains information about these types of customer calls.
Table B-8 shows the columns of the cust_calls table.

Table B-8. The cust_calls Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_num</td>
<td>INTEGER</td>
<td>Customer number (foreign key to customer table)</td>
</tr>
<tr>
<td>call_dtime</td>
<td>DATETIME YEAR TO MINUTE</td>
<td>Date and time call received</td>
</tr>
<tr>
<td>user_id</td>
<td>CHAR(18)</td>
<td>Name of person logging call (default is user login name)</td>
</tr>
<tr>
<td>call_code</td>
<td>CHAR(1)</td>
<td>Type of call (foreign key to call_type table)</td>
</tr>
<tr>
<td>call_descr</td>
<td>CHAR(240)</td>
<td>Description of call</td>
</tr>
<tr>
<td>res_dtime</td>
<td>DATETIME YEAR TO MINUTE</td>
<td>Date and time call resolved</td>
</tr>
<tr>
<td>res_descr</td>
<td>CHAR(240)</td>
<td>Description of how call was resolved</td>
</tr>
</tbody>
</table>

The customer, retail_customer, and wholesale_customer Tables

In this hierarchy, retail_customer and wholesale_customer are subtables that are created under the customer supertable, as Figure B-2 on page B-15 shows.

For information about table hierarchies, see the IBM Informix Database Design and Implementation Guide.

The customer Table: The customer table contains information about the retail stores that place orders from the distributor. Table B-9 on page B-7 shows the columns of the customer table.
### Table B-9. The customer Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_num</td>
<td>SERIAL</td>
<td>Unique customer identifier</td>
</tr>
<tr>
<td>customer_type</td>
<td>CHAR(1)</td>
<td>Code to indicate type of customer: R = retail, W = wholesale</td>
</tr>
<tr>
<td>customer_name</td>
<td>name_t</td>
<td>Name of customer</td>
</tr>
<tr>
<td>customer_loc</td>
<td>INTEGER</td>
<td>Location of customer (foreign key to location table)</td>
</tr>
<tr>
<td>contact_dates</td>
<td>LIST(DATETIME YEAR TO DAY NOT NULL)</td>
<td>Dates of contact with customer</td>
</tr>
<tr>
<td>cust_discount</td>
<td>percent</td>
<td>Customer discount</td>
</tr>
<tr>
<td>credit_status</td>
<td>CHAR(1)</td>
<td>Customer credit status: D = deadbeat, L = lost, N = new, P = preferred, R = regular</td>
</tr>
<tr>
<td>credit_num</td>
<td>CHAR(19)</td>
<td>Credit card number</td>
</tr>
<tr>
<td>expiration</td>
<td>DATE</td>
<td>Expiration data of credit card</td>
</tr>
</tbody>
</table>

#### The retail_customer Table:

The `retail_customer` table contains general information about retail customers. Table B-10 shows the columns of the `retail_customer` table.

### Table B-10. The retail_customer Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_num</td>
<td>SERIAL</td>
<td>Unique customer identifier</td>
</tr>
<tr>
<td>customer_type</td>
<td>CHAR(1)</td>
<td>Code to indicate type of customer: R = retail, W = wholesale</td>
</tr>
<tr>
<td>customer_name</td>
<td>name_t</td>
<td>Name of customer</td>
</tr>
<tr>
<td>customer_loc</td>
<td>INTEGER</td>
<td>Location of customer</td>
</tr>
<tr>
<td>contact_dates</td>
<td>LIST(DATETIME YEAR TO DAY NOT NULL)</td>
<td>Dates of contact with customer</td>
</tr>
<tr>
<td>cust_discount</td>
<td>percent</td>
<td>Customer discount</td>
</tr>
<tr>
<td>credit_status</td>
<td>CHAR(1)</td>
<td>Customer credit status: D = deadbeat, L = lost, N = new, P = preferred, R = regular</td>
</tr>
<tr>
<td>credit_num</td>
<td>CHAR(19)</td>
<td>Credit card number</td>
</tr>
<tr>
<td>expiration</td>
<td>DATE</td>
<td>Expiration data of credit card</td>
</tr>
</tbody>
</table>

#### The wholesale_customer Table:

The `wholesale_customer` table contains general information about wholesale customers. Table B-11 on page B-8 shows the columns of the `wholesale_customer` table.
Table B-11. The wholesale_customer Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>customer_num</td>
<td>SERIAL</td>
<td>Unique customer identifier</td>
</tr>
<tr>
<td>customer_type</td>
<td>CHAR(1)</td>
<td>Code to indicate type of customer:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R = retail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>W = wholesale</td>
</tr>
<tr>
<td>customer_name</td>
<td>name_t</td>
<td>Name of customer</td>
</tr>
<tr>
<td>customer_loc</td>
<td>INTEGER</td>
<td>Location of customer</td>
</tr>
<tr>
<td>contact_dates</td>
<td>LIST(DATETIME YEAR TO DAY NOT NULL)</td>
<td>Dates of contact with customer</td>
</tr>
<tr>
<td>cust_discount</td>
<td>percent</td>
<td>Customer discount</td>
</tr>
<tr>
<td>credit_status</td>
<td>CHAR(1)</td>
<td>Customer credit status:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D = deadbeat</td>
</tr>
<tr>
<td></td>
<td></td>
<td>L = lost</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N = new</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P = preferred</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R = regular</td>
</tr>
<tr>
<td>resale_license</td>
<td>CHAR(15)</td>
<td>Resale license number</td>
</tr>
<tr>
<td>terms_net</td>
<td>SMALLINT</td>
<td>Net term in days</td>
</tr>
</tbody>
</table>

The items Table

An order can include one or more items. One row exists in the items table for each item in an order. Table B-12 shows the columns of the items table.

Table B-12. The items Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>item_num</td>
<td>SMALLINT</td>
<td>Sequentially assigned item number for an order</td>
</tr>
<tr>
<td>order_num</td>
<td>INT8</td>
<td>Order number (foreign key to orders table)</td>
</tr>
<tr>
<td>stock_num</td>
<td>SMALLINT</td>
<td>Stock number for item (foreign key to stock table)</td>
</tr>
<tr>
<td>manu_code</td>
<td>CHAR(3)</td>
<td>Manufacturer code for item ordered (foreign key to stock table)</td>
</tr>
<tr>
<td>unit</td>
<td>CHAR(4)</td>
<td>Unit by which item is ordered (foreign key to stock table)</td>
</tr>
<tr>
<td>quantity</td>
<td>SMALLINT</td>
<td>Quantity ordered (value must be &gt; 1)</td>
</tr>
<tr>
<td>item_subtotal</td>
<td>MONEY(8,2)</td>
<td>Quantity ordered * unit price = total price of item</td>
</tr>
</tbody>
</table>

The location, location_non_us, and location_us Tables

In this hierarchy, location_non_us and location_us are subtables that are created under the location supertable, as shown in the diagram in “Table Hierarchies” on page B-14. For information about table hierarchies, see the IBM Informix Database Design and Implementation Guide.

The location Table

The location table contains general information about the locations (addresses) that the database tracks. Table B-13 on page B-9 shows the columns of the location table.
Table B-13. The location Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location_id</td>
<td>SERIAL</td>
<td>Unique identifier for location</td>
</tr>
<tr>
<td>loc_type</td>
<td>CHAR(2)</td>
<td>Code to indicate type of location</td>
</tr>
<tr>
<td>company</td>
<td>VARCHAR(20)</td>
<td>Name of company</td>
</tr>
<tr>
<td>street_addr</td>
<td>LIST(VARCHAR(25) NOT NULL)</td>
<td>Street address</td>
</tr>
<tr>
<td>city</td>
<td>VARCHAR(25)</td>
<td>City for address</td>
</tr>
<tr>
<td>country</td>
<td>VARCHAR(25)</td>
<td>Country for address</td>
</tr>
</tbody>
</table>

The location_non_us Table
The location_non_us table contains specific address information for locations (addresses) that are outside the United States. Table B-14 shows the columns of the location_non_us table.

Table B-14. The location_non_us Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location_id</td>
<td>SERIAL</td>
<td>Unique identifier for location</td>
</tr>
<tr>
<td>loc_type</td>
<td>CHAR(2)</td>
<td>Code to indicate type of location</td>
</tr>
<tr>
<td>company</td>
<td>VARCHAR(20)</td>
<td>Name of company</td>
</tr>
<tr>
<td>street_addr</td>
<td>LIST(VARCHAR(25) NOT NULL)</td>
<td>Street address</td>
</tr>
<tr>
<td>city</td>
<td>VARCHAR(25)</td>
<td>City for address</td>
</tr>
<tr>
<td>country</td>
<td>VARCHAR(25)</td>
<td>Country for address</td>
</tr>
<tr>
<td>province_code</td>
<td>CHAR(2)</td>
<td>Province code</td>
</tr>
<tr>
<td>zipcode</td>
<td>CHAR(9)</td>
<td>Zip code</td>
</tr>
<tr>
<td>phone</td>
<td>CHAR(15)</td>
<td>Phone number</td>
</tr>
</tbody>
</table>

The location_us Table
The location_us table contains specific address information for locations (addresses) that are in the United States. Table B-15 shows the columns of the location_us table.

Table B-15. The location_us Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>location_id</td>
<td>SERIAL</td>
<td>Unique identifier for location</td>
</tr>
<tr>
<td>loc_type</td>
<td>CHAR(2)</td>
<td>Code to indicate type of location</td>
</tr>
<tr>
<td>company</td>
<td>VARCHAR(20)</td>
<td>Name of company</td>
</tr>
<tr>
<td>street_addr</td>
<td>LIST(VARCHAR(25) NOT NULL)</td>
<td>Street address</td>
</tr>
<tr>
<td>city</td>
<td>VARCHAR(25)</td>
<td>City for address</td>
</tr>
<tr>
<td>country</td>
<td>VARCHAR(25)</td>
<td>Country for address</td>
</tr>
<tr>
<td>state_code</td>
<td>CHAR(2)</td>
<td>State code (foreign key to state table)</td>
</tr>
<tr>
<td>zip</td>
<td>CHAR(9)</td>
<td>Zip code</td>
</tr>
<tr>
<td>phone</td>
<td>CHAR(15)</td>
<td>Phone number</td>
</tr>
</tbody>
</table>
The manufact Table
Information about the manufacturers whose sporting goods are handled by the distributor is stored in the manufact table. Table B-16 shows the columns of the manufact table.

Table B-16. The manufact Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>manu_code</td>
<td>CHAR(3)</td>
<td>Manufacturer code</td>
</tr>
<tr>
<td>manu_name</td>
<td>VARCHAR(15)</td>
<td>Name of manufacturer</td>
</tr>
<tr>
<td>lead_time</td>
<td>INTERVAL DAY(3) TO DAY</td>
<td>Lead time for shipment of orders</td>
</tr>
<tr>
<td>manu_loc</td>
<td>INTEGER</td>
<td>Manufacturer location (foreign key to location table)</td>
</tr>
<tr>
<td>manu_account</td>
<td>CHAR(32)</td>
<td>Distributor account number with manufacturer</td>
</tr>
<tr>
<td>account_status</td>
<td>CHAR(1)</td>
<td>Status of account with manufacturer</td>
</tr>
<tr>
<td>terms_net</td>
<td>SMALLINT</td>
<td>Distributor terms with manufacturer (in days)</td>
</tr>
<tr>
<td>discount</td>
<td>percent</td>
<td>Distributor volume discount with manufacturer</td>
</tr>
</tbody>
</table>

The orders Table
The orders table contains information about orders placed by the customers of the distributor. Table B-17 shows the columns of the orders table.

Table B-17. The orders Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>order_num</td>
<td>SERIAL8(1001)</td>
<td>System-generated order number</td>
</tr>
<tr>
<td>order_date</td>
<td>DATE</td>
<td>Date order entered</td>
</tr>
<tr>
<td>customer_num</td>
<td>INTEGER</td>
<td>Customer number (foreign key to customer table)</td>
</tr>
<tr>
<td>shipping</td>
<td>ship_t</td>
<td>Special shipping instructions</td>
</tr>
<tr>
<td>backlog</td>
<td>BOOLEAN</td>
<td>Indicates order cannot be filled because the item is back ordered</td>
</tr>
<tr>
<td>po_num</td>
<td>CHAR(10)</td>
<td>Customer purchase order number</td>
</tr>
<tr>
<td>paid_date</td>
<td>DATE</td>
<td>Date order paid</td>
</tr>
</tbody>
</table>

The region Table
The region table contains information about the sales regions for the distributor. Table B-18 shows the columns of the region table.

Table B-18. The region Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>region_num</td>
<td>SERIAL</td>
<td>System-generated region number</td>
</tr>
<tr>
<td>region_name</td>
<td>VARCHAR(20)</td>
<td>Name of sales region</td>
</tr>
<tr>
<td>region_loc</td>
<td>INTEGER</td>
<td>Location of region office (foreign key to location table)</td>
</tr>
</tbody>
</table>

The sales_rep Table
The `sales_rep` table contains information about the sales representatives for the distributor. Table B-19 shows the columns of the `sales_rep` table.

Table B-19. The sales_rep Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rep_num</td>
<td>SERIAL(101)</td>
<td>System-generated sales rep number</td>
</tr>
<tr>
<td>name</td>
<td>name_t</td>
<td>Name of sales rep</td>
</tr>
<tr>
<td>region_num</td>
<td>INTEGER</td>
<td>Region in which sales rep works (foreign key to the region table)</td>
</tr>
<tr>
<td>home_office</td>
<td>BOOLEAN</td>
<td>Home office location of sales rep</td>
</tr>
<tr>
<td>sales</td>
<td>SET(ROW (month DATETIME YEAR TO MONTH, amount MONEY) NOT NULL)</td>
<td>Amount of monthly sales for rep</td>
</tr>
<tr>
<td>commission</td>
<td>percent</td>
<td>Commission rate for sales rep</td>
</tr>
</tbody>
</table>

The state Table
The `state` table contains the names and postal abbreviations, as well as sales tax information, for the 50 states of the United States. Table B-20 shows the columns of the `state` table.

Table B-20. The state Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>CHAR(2)</td>
<td>State code</td>
</tr>
<tr>
<td>sname</td>
<td>CHAR(15)</td>
<td>State name</td>
</tr>
<tr>
<td>sales_tax</td>
<td>percent</td>
<td>State sales tax</td>
</tr>
</tbody>
</table>

The stock Table
The `stock` table is a catalog of the items sold by the distributor. Table B-21 on page B-12 shows the columns of the `stock` table.
### Table B-21. The stock Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stock_num</td>
<td>SMALLINT</td>
<td>Stock number that identifies type of item</td>
</tr>
<tr>
<td>manu_code</td>
<td>CHAR(3)</td>
<td>Manufacturer code (foreign key to manufact)</td>
</tr>
<tr>
<td>unit</td>
<td>CHAR(4)</td>
<td>Unit by which item is ordered</td>
</tr>
<tr>
<td>description</td>
<td>VARCHAR(15)</td>
<td>Description of item</td>
</tr>
<tr>
<td>unit_price</td>
<td>MONEY(6,2)</td>
<td>Unit price</td>
</tr>
<tr>
<td>min_reord_qty</td>
<td>SMALLINT</td>
<td>Minimum reorder quantity</td>
</tr>
<tr>
<td>min_inv_qty</td>
<td>SMALLINT</td>
<td>Quantity of stock below which item should be reordered</td>
</tr>
<tr>
<td>manu_item_num</td>
<td>CHAR(20)</td>
<td>Manufacturer item number</td>
</tr>
<tr>
<td>unit_cost</td>
<td>MONEY(6,2)</td>
<td>Distributor cost per unit of item from manufacturer</td>
</tr>
<tr>
<td>status</td>
<td>CHAR(1)</td>
<td>Status of item: A = active, D = discontinued, N = no order</td>
</tr>
<tr>
<td>bin_num</td>
<td>INTEGER</td>
<td>Bin number</td>
</tr>
<tr>
<td>qty_on_hand</td>
<td>SMALLINT</td>
<td>Quantity in stock</td>
</tr>
<tr>
<td>bigger_unit</td>
<td>CHAR(4)</td>
<td>Stock unit for next larger unit (for same stock_num and manu_code)</td>
</tr>
<tr>
<td>per_bigger_unit</td>
<td>SMALLINT</td>
<td>How many of this item in bigger_unit</td>
</tr>
</tbody>
</table>

### The stock_discount Table

The stock_discount table contains information about stock discounts. (There is no primary key.) Table B-22 shows the columns of the stock_discount table.

### Table B-22. The stock_discount Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>discount_id</td>
<td>SERIAL</td>
<td>System-generated discount identifier</td>
</tr>
<tr>
<td>stock_num</td>
<td>SMALLINT</td>
<td>Distributor stock number (part of foreign key to stock table)</td>
</tr>
<tr>
<td>manu_code</td>
<td>CHAR(3)</td>
<td>Manufacturer code (part of foreign key to stock table)</td>
</tr>
<tr>
<td>unit</td>
<td>CHAR(4)</td>
<td>Unit by which item is ordered (each, pair, case, and so on) (foreign key to units table; part of foreign key to stock table)</td>
</tr>
<tr>
<td>unit_discount</td>
<td>percent</td>
<td>Unit discount during sale period</td>
</tr>
<tr>
<td>start_date</td>
<td>DATE</td>
<td>Discount start date</td>
</tr>
<tr>
<td>end_date</td>
<td>DATE</td>
<td>Discount end date</td>
</tr>
</tbody>
</table>

### The units Table

The units table contains information about the units in which the inventory items can be ordered. Each item in the stock table is available in one or more types of container. Table B-23 on page B-13 shows the columns of the units table.
### User-Defined Routines and Extended Data Types

The **superstores_demo** database uses *user-defined routines* (UDRs) and extended data types.

A UDR is a routine that you define that can be invoked within an SQL statement or another UDR. A UDR can either return values or not.

The data type system of Dynamic Server is an extensible and flexible system that supports the creation of following kinds of data types:

- Extensions of existing data types by, redefining some of the behavior for data types that the database server provides.
- Definitions of customized data types by a user.

This section lists the extended data types and UDRs created for the **superstores_demo** database. For information about creating and using UDRs and extended data types, see *IBM Informix User-Defined Routines and Data Types Developer’s Guide*.

The **superstores_demo** database creates the *distinct* data type, percent, in a UDR, as follows:

*CREATE DISTINCT TYPE percent AS DECIMAL(5,5)*;

*DROP CAST (DECIMAL(5,5) AS percent)*;

*CREATE IMPLICIT CAST (DECIMAL(5,5) AS percent)*;

The **superstores_demo** database creates the following named row types:

- **location** hierarchy:
  - location_t
  - loc_us_t
  - loc_non_us_t
- **customer** hierarchy:
  - name_t
  - customer_t
  - retail_t
  - whlsale_t
- **orders** table
  - ship_t

#### Location_t Definition

<table>
<thead>
<tr>
<th>column</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>location_id</td>
<td>SERIAL</td>
</tr>
<tr>
<td>loc_type</td>
<td>CHAR(2)</td>
</tr>
<tr>
<td>company</td>
<td>VARCHAR(20)</td>
</tr>
<tr>
<td>street_addr</td>
<td>LIST(VARCHAR(25) NOT NULL)</td>
</tr>
<tr>
<td>city</td>
<td>VARCHAR(25)</td>
</tr>
<tr>
<td>country</td>
<td>VARCHAR(25)</td>
</tr>
</tbody>
</table>

---

Table B-23. The units Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unit_name</td>
<td>CHAR(4)</td>
<td>Units by which an item is ordered (each, pair, case, box)</td>
</tr>
<tr>
<td>unit_descr</td>
<td>VARCHAR(15)</td>
<td>Description of units</td>
</tr>
</tbody>
</table>
loc_us_t definition
state_code CHAR(2)
zip ROW(code INTEGER, suffix SMALLINT)
phone CHAR(18)

loc_non_us_t definition
province_code CHAR(2)
zipcode CHAR(9)
phone CHAR(15)

name_t definition
first VARCHAR(15)
last VARCHAR(15)

customer_t definition
customer_num SERIAL
customer_type CHAR(1)
customer_name name_t
customer_loc INTEGER
contact_dates LIST(DATETIME YEAR TO DAY NOT NULL)
cust_discount percent
credit_status CHAR(1)

retail_t definition
credit_num CHAR(19)
expiration DATE

whlsale_t definition
resale_license CHAR(15)
terms_net SMALLINT

ship_t definition
date DATE
weight DECIMAL(8,2)
charge MONEY(6,2)
instruct VARCHAR(40)

Table Hierarchies
Figure B-2 on page B-15 shows how the hierarchical tables of the superstores_demo database are related. See "The customer and location Tables" on page B-16 for an explanation of the foreign key and primary relationships between those two tables, which are indicated by shaded arrows that point from the customer.custnum and customer.loc columns to the location.location_id columns in the following diagram.
Referential Relationships

The tables of the superstores_demo database are linked by the primary-foreign key relationships that are identified in this section. This type of relationship is called a referential constraint because a foreign key in one table references the primary key in another table.

The customer and orders Tables
The customer table contains a customer_num column that holds a number that identifies a customer. The orders table also contains a customer_num column that stores the number of the customer who placed a particular order. In the orders table, the customer_num column is a foreign key that references the customer_num column in the customer table.

The orders and items Tables
The orders and items tables are linked by an order_num column that contains an identification number for each order. If an order includes several items, the same order number appears in several rows of the items table. In the items table, the order_num column is a foreign key that references the order_num column in the orders table.

The items and stock Tables
The items table and the stock table are joined by three columns: the stock_num column, which stores a stock number for an item, the manu_code column, which stores a code that identifies the manufacturer, and the units column, which identifies the types of unit in which the item can be ordered. You need the stock number, the manufacturer code, and the units to uniquely identify an item. The same stock number and manufacturer code can appear in more than one row of the items table, if the same item belongs to separate orders. In the items table, the stock_num, manu_code, and unit columns are foreign keys that reference the stock_num, manu_code, and unit columns in the stock table.

The stock and catalog Tables
The stock table and catalog table are joined by three columns: the stock_num column, which stores a stock number for an item, the manu_code column, which stores a code that identifies the manufacturer, and the unit column, which identifies the type of units in which the item can be ordered. You need all three columns to uniquely identify an item. In the catalog table, the stock_num,
**manu_code**, and **unit** columns are foreign keys that reference the **stock_num**, **manu_code**, and **unit** columns in the **stock** table.

**The stock and manufact Tables**
The **stock** table and the **manufact** table are joined by the **manu_code** column. The same manufacturer code can appear in more than one row of the **stock** table if the manufacturer produces more than one piece of equipment. In the **stock** table, the **manu_code** column is a foreign key that references the **manu_code** column in the **manufact** table.

**The cust_calls and customer Tables**
The **cust_calls** table and the **customer** table are joined by the **customer_num** column. The same customer number can appear in more than one row of the **cust_calls** table if the customer calls the distributor more than once with a problem or question. In the **cust_calls** table, the **customer_num** column is a foreign key that references the **customer_num** column in the **customer** table.

**The call_type and cust_calls Tables**
The **call_type** and **cust_calls** tables are joined by the **call_code** column. The same call code can appear in more than one row of the **cust_calls** table, because many customers can have the same type of problem. In the **cust_calls** table, the **call_code** column is a foreign key that references the **call_code** column in the **call_type** table.

**The state and customer Tables**
The **state** table and the **customer** table are joined by a column that contains the state code. This column is called **code** in the **state** table and **state** in the **customer** table. If several customers live in the same state, the same state code appears in several rows of the table. In the **customer** table, the **state** column is a foreign key that references the **code** column in the **state** table.

**The customer and location Tables**
In the **customer** table, the **customer_loc** column is a foreign key that references the **location_id** of the **location** table. The **customer_loc** and **location_id** columns each uniquely identify the customer location.

**The manufact and location Tables**
The **manu_loc** column in the **manufact** table is a foreign key that references the **location_id** column, which is the primary key in the **location** table. Both **manu_loc** and **location_id** uniquely identify the manufacturer location.

**The state and location_us Tables**
The **state** and **location_us** tables are joined by the column that contains the state code. The **state_code** column in the **location_us** table is a foreign key that references the **code** column in the **state** table.

**The sales_rep and region Tables**
The **region_num** column is the primary key in the **region** table. It is a system-generated region number. The **region_num** column in the **sales_rep** table is a foreign key that references and joins the **region_num** column in the **region** table.

**The region and location Tables**
The **region_loc** column in the **region** table identifies the regional office location. It is a foreign key that references the **location_id** column in the **location** table, which is a unique identifier for location.
The stock and stock_discount Tables
The stock table and the stock_discount table are joined by three columns: stock_num, manu_code, and unit. These columns form the primary key for the stock table. The stock_discount table has no primary key and references the stock table.

The stock and units Tables
The unit_name column of the units table is a primary key that identifies the kinds of units that can be ordered, such as case, pair, box, and so on. The unit column of the stock table joins the unit_name column of the units table.
Appendix C. Accessibility

IBM strives to provide products with usable access for everyone, regardless of age or ability.

Accessibility features for IBM Informix Dynamic Server

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

Accessibility Features

The following list includes the major accessibility features in IBM Informix Dynamic Server. These features support:
- Keyboard-only operation.
- Interfaces that are commonly used by screen readers.
- The attachment of alternative input and output devices.

Tip: The IBM Informix Dynamic Server Information Center and its related publications are accessibility-enabled for the IBM Home Page Reader. You can operate all features using the keyboard instead of the mouse.

Keyboard Navigation

This product uses standard Microsoft Windows navigation keys.

Related Accessibility Information

IBM is committed to making our documentation accessible to persons with disabilities. Our publications are available in HTML format so that they can be accessed with assistive technology such as screen reader software. The syntax diagrams in our publications are available in dotted decimal format. For more information about the dotted decimal format, go to "Dotted Decimal Syntax Diagrams."

You can view the publications for IBM Informix Dynamic Server in Adobe Portable Document Format (PDF) using the Adobe Acrobat Reader.

IBM and Accessibility

See the IBM Accessibility Center at [http://www.ibm.com/able](http://www.ibm.com/able) for more information about the commitment that IBM has to accessibility.

Dotted Decimal Syntax Diagrams

The syntax diagrams in our publications are available in dotted decimal format, which is an accessible format that is available only if you are using a screen reader.

In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), the elements can appear on the same line, because they can be considered as a single compound syntax element.
Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that your screen reader is set to read punctuation. All syntax elements that have the same dotted decimal number (for example, all syntax elements that have the number 3.1) are mutually exclusive alternatives. If you hear the lines 3.1 USERID and 3.1 SYSTEMID, your syntax can include either USERID or SYSTEMID, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, the word or symbol is preceded by the backslash (\) character. The * symbol can be used next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is read as 3 \* FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3\* FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol that provides information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, this identifies a reference that is defined elsewhere. The string following the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you should refer to a separate syntax fragment OP1.

The following words and symbols are used next to the dotted decimal numbers:

? Specifies an optional syntax element. A dotted decimal number followed by the ? symbol indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element (for example, 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that syntax elements NOTIFY and UPDATE are optional; that is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! Specifies a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicates that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the same dotted decimal number can specify a ! symbol. For example, if you hear the lines
2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In this example, if you include the FILE keyword but do not specify an option, default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP only applies to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

* Specifies a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data-area, you know that you can include more than one data area or you can include none. If you hear the lines 3*, 3 HOST, and 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

**Notes:**
1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.

2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you could write HOST STATE, but you could not write HOST HOST.

3. The * symbol is equivalent to a loop-back line in a railroad syntax diagram.

+ Specifies a syntax element that must be included one or more times. A dotted decimal number followed by the + symbol indicates that this syntax element must be included one or more times. For example, if you hear the line 6.1+ data-area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. As for the * symbol, you can only repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loop-back line in a railroad syntax diagram.
Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created
programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation  
J46A/G4  
555 Bailey Avenue  
San Jose, CA 95141-1003  
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM’s suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM’s application programming interfaces.
Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. (enter the year or years). All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

**Trademarks**

The following terms are used in one or more IBM Informix product publications and the terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

<table>
<thead>
<tr>
<th>Term</th>
<th>Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>OS/2</td>
</tr>
<tr>
<td>AnalystStudio</td>
<td>OS/390</td>
</tr>
<tr>
<td>C-ISAM</td>
<td>OS/400</td>
</tr>
<tr>
<td>Cloudscape</td>
<td>Passport Advantage</td>
</tr>
<tr>
<td>DataBlade</td>
<td>ProjectConsole</td>
</tr>
<tr>
<td>DB2</td>
<td>Rational</td>
</tr>
<tr>
<td>DB2 Connect</td>
<td>Rational Suite</td>
</tr>
<tr>
<td>DB2 Universal Database</td>
<td>Rational Unified Process</td>
</tr>
<tr>
<td>Distributed Relational Database Architecture</td>
<td>RedBack</td>
</tr>
<tr>
<td>Domino</td>
<td>Red Brick</td>
</tr>
<tr>
<td>DRDA</td>
<td>RequisitePro</td>
</tr>
<tr>
<td>Dynamic Connect</td>
<td>RETAIN</td>
</tr>
<tr>
<td>IBM</td>
<td>SQL/DS</td>
</tr>
<tr>
<td>Informix</td>
<td>SystemBuilder</td>
</tr>
<tr>
<td>Lotus</td>
<td>Tivoli</td>
</tr>
<tr>
<td>Lotus Notes</td>
<td>TME 10</td>
</tr>
<tr>
<td>MQSeries</td>
<td>UniData</td>
</tr>
<tr>
<td>NetView</td>
<td>UniVerse</td>
</tr>
<tr>
<td>Notes</td>
<td>WebSphere</td>
</tr>
<tr>
<td>NUMA-Q</td>
<td>z/OS</td>
</tr>
</tbody>
</table>

Adobe, Acrobat, Portable Document Format (PDF), and PostScript are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.
Index

Special characters

( _ ), underscore
in SQL identifiers 3-35
( ), comma
list separator 3-49, 3-61
( : ), colon
cast ( :: ) operator 2-43, 2-46
DATETIME delimiter 2-11
INTERVAL delimiter 2-18
list separator 3-29, 3-35, 3-49, 3-57, 3-61
(! ), not equal to
relational operator 2-46
( / ), slash
DATE separator 2-9, 2-37, 3-21
division operator 2-34, 2-46
pathname delimiter 3-7, 3-27, 3-57
( ’ ), single quotes
string delimiter 3-24, 3-35
( “ ), double quotes
delimited SQL identifiers 3-35
string delimiter 2-1, 2-19, 2-21, 2-28
( [ ] ), parentheses
delimiters in expressions 2-37
( $ ), dollar sign
currency symbol 2-20, 3-24
pathname indicator 3-14, 3-61
( \ ), backslash
invalid as delimiter 3-22
pathname delimiter 3-9, 3-52
( { } ), braces
MATCHES range delimiters 2-31
substring operator 2-6, 2-29, 2-46
( % ), percentage
DBTIME escape symbol 3-31
pathname indicator 3-14, 3-30
( > ), greater than
angle ( < > ) brackets 2-7, A-17
relational operator 1-7, 2-46
( < ), less than
angle ( < > ) brackets 2-7, A-17
relational operator 2-46, 3-22
( : ), vertical bar
absolute value delimiter 2-16
concatenation ( ( ) ) operator 2-46
field delimiter 3-22
( # ), sharp
comment indicator 3-4
( [ ] ), braces
collection delimiters 2-19, 2-21, 2-27, 2-28, 2-39
pathname delimiters 3-6
( - ), hyphen
DATE separator 3-20
DATETIME delimiter 2-11
INTERVAL delimiter 2-18
subtraction operator 2-34, 2-46
symbol in syscoluma 1-5, 1-16
symbol in sysfragauth 1-27
symbol in systabauth 1-47
unary operator 2-35, 2-46
( ), comma
decimal point 3-24

( ), comma (continued)
list separator 2-21, 2-24, 3-29
thousands separator 2-20
( ), period
DATE separator 3-21
DATETIME delimiter 2-11
decimal point 2-14, 2-20, 3-24
evaluation symbol 3-5
INTERVAL delimiter 2-18
membership operator 2-46
nested dot notation 2-40
( ), blank space
DATETIME delimiter 2-11
INTERVAL delimiter 2-18
padding CHAR values 2-7
padding VARCHAR values 2-30
( * ), asterisk
multiplication operator 2-26, 2-34, 2-37, 2-46
systabauth value 1-5, 1-47
wildcard symbol 1-15, 1-37
(+), plus sign
addition operator 2-34, 2-46
truncation indicator 3-42
unary operator 2-46
( = ), equality
assignment operator 3-9
relational operator 1-15, 2-5, 2-9, 2-46
( - ), tildes
pathname indicator 3-7

A
Abbreviated year values 2-12, 3-18, 3-19, 3-21, 3-32
AC_CONFIG environment variable 3-32
ac_config.std file 3-14
ACCESS keyword 1-11, 2-33
Access method
B-tree 1-13, 1-31, 3-34
built-in 1-11, 1-13
primary 1-11, 1-46
R-Tree 3-34
secondary 1-11, 1-21, 1-32, 2-22
sysams data 1-11
syindices data 1-32
sysoptclasses data 1-34
systabamdata data 1-46
accessibility C-1
keyboard C-1
shortcut keys C-1
Accessibility
dotted decimal format of syntax diagrams C-1
syntax diagrams, reading in a screen reader C-1
Activity-log files 3-55
Addition (+) operator 2-34, 2-46
Administrative listener port 3-46
AFCRASH configuration parameter 3-34
AFDEBUG environment variable 3-34
Aggregate functions
built-in 2-19, 2-21, 2-27
no BYTE argument 2-6
no collection arguments 2-19, 2-21, 2-27

© Copyright IBM Corp. 1996, 2007

X-1
Compiling (continued)

multithreaded ESQL/C applications 3-67
Complex data type 2-38, 2-40
  collection types 2-39
  ROW types 2-40
  sysattrtypes data 1-13

Compliance
  ANSI/ISO standard for SQL 1-57, 3-17
  sql_languages.conformance 1-59
  X/Open CAE standards 1-57
  XPG4 standard 1-58

Concatenation
  (1) operator 2-46
  concsm.cfg file 3-47

Configuration file
  .cshrc file 3-5
  .informix file 3-4, 3-7, 3-36, 3-40
  .profile file 3-5
  .login file 3-5
  for communications support module 3-47, 3-49
  for connectivity 3-46, 3-50, 3-52
  for database servers 3-35, 3-58
  for High-Performance Loader 3-62
  for MaxConnect 3-46
  for ON-Bar utility 3-14
  for onxfer utility 3-68
  for terminal I/O 3-52, 3-67

Configuration parameters
  COSERVER 3-58
  DBSPACETEMP 3-30
  DEF_TABLE_LOCKMODE 3-40
  DIRECTIVES 3-40
  DISABLE_B162428_XA_FIX 3-45
  END 3-58
  EXT_DIRECTIVES 1-23, 3-41
  MTRACEx_OFF 1-50, 1-51
  NODE 3-58
  OPCACHEMAX 3-50
  OPT_GOAL 3-60
  OPTCOMPIND 3-59
  shared memory base 3-45
  STACKSIZE 3-52
  STMT_CACHE 3-66
  TABLESPACE 3-30
  USEOSTIME 2-12
  CONNECT DEFAULT statement 3-51
  Connect privilege 1-7, 1-52
  CONNECT statement 3-26, 3-48, 3-51

Connections
  authentication 3-49
  coserver 3-51
  INFORMIXCONRETRY environment variable 3-47
  INFORMIXCONTIME environment variable 3-48
  INFORMIXSERVER environment variable 3-50
  Connectivity information 3-46, 3-51

Constraints (continued)

  check
    creation-time value 3-22
    loading performance B-2
    syschecks data 1-15
    syscheckdrep data 1-15
    syscoldepend data 1-17
  column
    sysconstraints data 1-21
  not null
    collection data types 2-19, 2-21, 2-27, 2-39
    syscoldepend data 1-17

Constraints (continued)

  not null (continued)
    syscolumns data 1-19
    sysconstraints data 1-21
    object mode 1-33
    primary key
      sysconstraints data 1-21
      sysreferences data 1-41
    unique SERIAL values 2-25
    unique SERIAL8 values 2-26
  referential
    stores_demo data A-5
    superstores_demo data B-15
    sysconstraints data 1-21
    sysreferences data 1-41
    table
      sysconstraints data 1-21
      unique
        sysconstraints data 1-21
        violations 1-53
  Constructors 2-27, 2-39

Converting data types
  CHAR and NCHAR 3-25
  DATE and DATETIME 2-43
  INTEGER and DATE 2-43
  number and string 2-42
  number to number 2-42
  retyping a column 2-41
  VARCHAR and NVARCHAR 3-25

Coserver
  sysexternal data 1-27
  sysviolations data 1-53

COSERVER configuration parameter 3-51, 3-58

CPFIRST environment variable 3-15

CPU cost 3-65

CREATE ACCESS METHOD statement 1-11
CREATE CAST statement 1-14, 2-43
CREATE DATABASE statement 3-26
CREATE DISTINCT TYPE statement 1-56, 2-15, B-13
CREATE DUPLICATE statement 1-29
CREATE EXTERNAL TABLE statement 1-26, 1-27
CREATE FUNCTION statement 1-42
CREATE IMPLICIT CAST statement B-13
CREATE INDEX statement 1-30, 1-31, 1-33, 1-41, 1-49, 3-34
  storage options 3-34
CREATE OPAQUE TYPE statement 2-22
CREATE OPERATOR CLASS statement 1-34
CREATE OPTICAL CLUSTER statement 1-36
CREATE PROCEDURE statement 1-42, 3-58
CREATE ROLE statement 1-42
CREATE ROUTINE FROM statement 1-42, 3-58
CREATE ROW TYPE statement 1-18, 2-23
CREATE SCHEMA statement 1-3
CREATE SEQUENCE statement 1-45
CREATE SYNONYM statement 1-45, 1-46
CREATE TABLE statement
  assigning data types 2-3
  default lock mode 3-40
  default privileges 3-58
  SET constructor 2-27
  typed tables 2-23
CREATE TEMP TABLE statement 3-29
CREATE TRIGGER statement 1-52
CREATE VIEW statement 1-5, 1-53

Credential 3-50

Currency symbol 2-20, 3-24

Current date 1-22, 3-18
CURRENT keyword 2-34, 3-37

cust_calls table in stores_demo database A-4
cust_calls table in superstores_demo database B-6
customer table in sales_demo database B-3
customer table in stores_demo database A-1
customer table in superstores_demo database B-6, B-7

D

Data compression 3-54
Data corruption 1-8, 1-16
Data dependencies
syscheckudrdep data 1-15
syscoldepend data 1-17
sysdepend data 1-23
sysnewdepend data 1-33
Data dictionary 1-2
Data distributions 1-8, 1-24, 3-33
Data encryption 3-55
Data integrity 1-59
Data pages 1-16, 1-31, 1-48
Data types
approximate 1-58
BLOB 2-5
BOOLEAN 2-5
BYTE 2-6
casting 2-41, 2-45
CHAR 2-7
CHARACTER 2-5, 2-8
CHARACTER VARYING 2-8
classified by category 2-2
CLOB 2-8
collection 2-39
complex 2-38
conversion 2-41
DATE 2-9
DATETIME 2-10
DEC 2-12
DECIMAL 2-12
distinct 2-14, 2-40
DISTINCT 2-14
DOUBLE PRECISION 2-15
effect numeric 1-58
extended 2-38
fixed point 2-13
FLOAT 2-15
floating-point 2-12, 2-15, 2-28
IDSECURITYLABEL 2-15
inheritance 2-23
INT 2-16
INT8 2-16
INTEGER 2-16
internal 2-5
INTERVAL 2-16
LIST 2-18
LVARCHAR 2-19
MONEY 2-20
MULTISET 2-21
named ROW 2-23
NCHAR 2-21
NUMERIC 2-22
NVARCHAR 2-22
opaque 2-22, 2-41
OPAQUE 2-22
REAL 2-22
ROW 2-23, 2-24
sequential integer 2-26

Data types (continued)
SERIAL 2-25
SERIAL8 2-26
SET 2-27
simple large object 2-32
SMALLFLOAT 2-28
SMALLINT 2-28
smart large object 2-32
summary list 2-3
TEXT 2-28
unique numeric value 2-26
unnamed ROW 2-24
VARCHAR 2-30

Data warehousing B-1
Database identifiers 3-35
Database server administrator (DBSA) 1-3
Database Server Administrator (DBSA) 3-53
Database servers
attributes in Information Schema view 1-59
codset 1-60
coserver name 3-51
default connection 3-50
default isolation level 1-60
optimizing queries 3-60
pathname for 3-26
remote 3-15, 3-36
role separation 3-53
server name 1-22, 3-27

DATABASE statement 3-26

Databases
data types 2-2
dimensional B-2
identifiers 3-34
joins in stores_demo A-5
object-relational B-2
objects, sysobjectstate data 1-33
privileges 1-52
sales_demo B-1
stores_demo A-1
superstores_demo B-2, B-4
syscrd 1-3
sysmaster 1-3
sysutils 1-3
sysuuid 1-3

DataBlade modules
Client and Server API 2-33
data types (sysbuiltintypes) 1-3
trace messages (sysstracemsgs) 1-50
user messages (syserrors) 1-25

DATE data type
abbreviated year values 3-18
casting to integer 2-43
converting to DATETIME 2-43
defined 2-9
display format 3-20
in expressions 2-33, 2-35
international date formats 2-9
source data 2-36

DATE() function 2-36, 3-21

DATETIME data type
abbreviated year values 3-18
converting to DATE 2-43
defined 2-10
display format 3-31
EXTEND function 2-35
extending precision 2-34
field qualifiers 2-10
DATETIME data type (continued)
  in expressions 2-33, 2-37
  international formats 2-10, 2-12, 2-17
  length (syscolumns) 1-20
  literal values 2-11
  precision and size 2-10
  source data 2-36
  two-digit year values and DBDATE variable 2-12
  year to fraction example 2-11

DAY keyword
  DATETIME qualifier 2-10
  INTERVAL qualifier 2-17
  UNITS operator 2-9, 2-36
  DB-Access utility 1-7, 1-57, 3-8, 3-23, 3-26, 3-30, 3-50
  DBA privilege 1-25, 1-50, 1-51, 1-52
  DBA routines 1-39
  DBACOINOIGN environment variable 3-16
  DBANSIWARN environment variable 3-17
  DBBLOBBUF environment variable 3-17
  DBCENTURY environment variable
     defined 3-18
     effect on functionality of DBDATE 3-21
     expanding abbreviated years 2-12, 3-18
  DBDATE environment variable 2-9, 3-20
  DBDELMINER environment variable 3-22
  DBEDIT environment variable 3-22
  dbexport utility 3-22
  DBFLTMASK environment variable 3-23
  DBLANG environment variable 3-23
  dbload utility 2-5, 2-6, 2-29, 3-22
  DBMONEY environment variable 2-20, 3-24
  DBNLS environment variable 3-25
  DBNPLOAD environment variable 3-26
  DBPATH environment variable 3-26
  DBPRINT environment variable 3-28
  DBREMOCEMD environment variable 3-28
  dbschema utility 3-68
  Dbserv group 3-51
  DBSERVERNAME configuration parameter 3-51
  dbservername.cmd batch file 3-9
  dbslice 1-28, 1-29
  dbspace
     for BYTE or TEXT values 1-14
     for system catalog 1-2
     for table fragments 1-27
     for temporary tables 3-29
     name 3-35
  DBSPACE keyword 1-28
  DBSPACETEMP configuration parameter 3-29
  DBSPACETEMP environment variable 3-29
  DBTEMP environment variable 3-30
  DBTIME environment variable 2-12, 3-31
  DBUPSPACE environment variable 3-33
  DCE-GSS communications support module (CSM) 3-49

DECIMAL data type
  built-in casts 2-42
  defined 2-12
  disk storage 2-13
  display format 3-23, 3-24
  fixed point 2-13
  floating point 2-13
  length (syscolumns) 1-20
  Decimal digits, display of 3-23
  Decimal point
     DBFLTMASK setting 3-23
     DBMONEY setting 3-24
     DECIMAL radix 2-14

Decimal separator 3-24
DECLARE statement 3-60
DECRYPT_BINARY function 2-9
DECRYPT_CHAR function 2-9
DEF_TABLE_LOCKMODE configuration parameter 3-40
DEFAULT_ATTACH environment variable 3-34

Defaults
  C compiler 3-47
  century 3-18, 3-32
  CHAR length 2-7
  character set for SQL identifiers 3-35
  compilation order 3-15
  configuration file 3-58
  connection 3-50
  data type 2-24
  database server 3-27, 3-50
  DATE display format 2-9
  DATE separator 3-21
  DATETIME display format 2-12
  DECIMAL precision 2-13
  detached indexes 3-34
  detail level 3-55
  disk space for sorting 3-33
  fetch buffer size 3-36
  heap size 3-56
  isolation level 1-60
  join method 3-59
  level of parallelism 3-62
  lock mode 3-40
  message directory 3-23
  MONEY scale 2-20
  operator class 1-11, 1-34
  printing program 3-28
  query optimizer goal 3-60
  sysdefaults.default 1-22
  table privileges 3-58
  temporary dbspace 3-30
  termcap file 3-67
  text editor 3-22
  DEFINE statement of SPL 2-25, 2-26
  Delete privilege 1-28, 1-47, 3-58
  DELETE statements 1-8, 1-53
  Delete trigger 1-52
  DELIMIDENT environment variable 3-34
  DELIMITED files 1-26, 1-27
  Delimited identifiers 3-34, 3-35
  Delimiter
     for DATETIME values 2-11
     for fields 1-27, 3-22
     for identifiers 3-34
     for INTERVAL values 2-18
  Demonstration databases
     tables A-1, A-4, B-5
     Descending index 1-31
  DESCRIBE statement 3-44
  Describe-for-updates 3-45
  destroy() support function 2-39
  Detached index 3-34
  Deutsche mark (DM) currency symbol 3-25
  Diagnostics table 1-53
  Dimension tables, in push-down hash joins 3-39
  DIRECTIVES configuration parameters 3-40
  Directives for query optimization 3-40, 3-59, 3-60
  Disabilities, visual
     reading syntax diagrams C-1
  disability C-1
  Disabled object 1-53
Disk space
  for data distributions 3-33
  for temporary data 3-30
Distinct data types
casts 2-44
  defined 2-14
  sysxtddesc data 1-54
  sysxtdtypes data 1-55, 1-56, 2-15
Distributed Computing Environment (DCE) 3-67, 3-68
Distributed queries 2-38, 3-15, 3-36
Dollar ($) sign 2-20, 3-24
Dotted decimal format of syntax diagrams C-1
double (C) data type 2-15
Double-precision floating-point number 2-15
DROP CAST statement B-13
DROP DATABASE statement 3-26
DROP FUNCTION statement 1-39
DROP INDEX statement 1-49
DROP OPTICAL CLUSTER statement 1-36
DROP PROCEDURE statement 1-39
DROP ROUTINE statement 1-39
DROP ROW TYPE statement 2-23
DROP SEQUENCE statement 3-68
DROP TABLE statement 3-68
DROP TYPE statement 2-15, 2-22
DROP VIEW statement 1-57, 3-68

E
EBCDIC collation 1-27, 1-60
Editor, DBEDIT setting 3-22
EMACS text editor 3-23
Empty set 2-29
ENCRYPTDES function 2-8
ENCRYPTDES function 2-8
Encryption 3-55
END configuration parameter 3-58
Enterprise Replication 1-3
env utility 3-6
ENVIGNORE environment variable
defined 3-5, 3-35
relation to chkenv utility 3-7
Environment configuration file
debbuging with chkenv 3-7
setting environment variables in UNIX 3-4
Environment variables 3-4
AC_CONFIG 3-14
AFDEBUG 3-14
ANSIOWNER 3-14
CC_ZEROLEVEL 3-11
CLIENT_LOCALE 3-11, 3-21
command-line utilities 3-8
CPFIRST 3-15
DB_LOCALE 3-11
DBACCCNOIGN 3-16
DBANSIWARN 3-17
DBBLOBBUF 3-17
DBCENTURY 3-18
DBDATE 2-9, 3-20
DBDELIMITER 3-22
DBEDIT 3-22
DBFLT_MASK 3-23
DBLANG 3-23
DBMONEY 2-20, 3-24
DBNILS 3-25
DBONUPLOAD 3-26
DBPATH 3-26

Environment variables (continued)
DBPRINT 3-28
DBREMO_TECMD 3-28
DBSPACETEMP 3-29
DBTEMP 3-30
DBTIME 2-12, 3-31
DBUPSPACE 3-33
DEFAULT_ATTACH 3-34
DELIMIDENT 3-34
displaying current settings 3-6, 3-9
ENVIGNORE 3-35
ESQLMF 3-11
FET_BUF_SIZE 3-15, 3-36
GL_DATE 2-9, 3-20
GL_DATETIME 2-12, 3-20
GLOBAL_DETACH_INFORM 3-11, 3-36
GLS8BITSYS 3-11
how to set
  in Bourne shell 3-5
  in C shell 3-5
  in Korn shell 3-5
how to set in Bourne shell 3-5
how to set in Korn shell 3-5
IBM_XPS_PARAMS 3-37
IFMX_CART_ALRM 3-38
IFMX_OPT_NON_DIM_TABS 3-39
IFX_DEF_TABLE_LOCKMODE 3-40
IFX_DIRECTIVES 3-40
IFX_EXTDIRECTIVES 1-23, 3-41
IFX_LONGID 3-41
IFX_NETBUF_PVTPOOL_SIZE 3-42
IFX_NETBUF_SIZE 3-42
IFX_NO_LRU_TUNING 3-43
IFX_NO_TIMELIMIT_WARNING 3-43
IFX_NODBPROC 3-43
IFX_ONTAPE_FILE_PREFIX 3-44
IFX_OPT_FACT_TABS 3-38
IFX_PAD_VARCHAR 3-44
IFX_UPDDESC 3-44
IFX_XASTDCOMPLIANCE_XAEND 3-45
IFX_XFER_SHMBASE 3-45
IMADMIN 3-45
IMCONFIG 3-46
IMCADMIN 3-46
INF_ROLE_SEP 3-53
INFORMIX 3-47
INFORMIXCONCSMCFG 3-47
INFORMIXCONCRETRY 3-47
INFORMIXCONTIME 3-48
INFORMIXCPPMAP 3-49
INFORMIXDIR 3-49
INFORMIXKEYTAB 3-49
INFORMIXOPCACHE 3-50
INFORMIXSERVER 3-50
INFORMIXSHMBASE 3-51
INFORMIXSQLHOSTS 3-51
INFORMIXSTACKSIZE 3-52
INFORMIXTERM 3-52
INTERACTIVE_DESKTOP_OFF 3-54
ISM_COMPRESSION 3-54
ISM_DEBUG_INFO 3-54
ISM_DEBUG_LEVEL 3-55
ISM_DEBUG 3-55
ISM_ENCRYPTION 3-55
ISM_MAXLOGSIZE 3-55
ISM_MAXLOGVERS 3-55
JAR_TEMP_PATH 3-56
JAVA_COMPILER 3-56

Index  X-7
Explicit
Explicit
Explicit
Executable
Exact

Environment variables (continued)
JVM_MAX_HEAP_SIZE 3-56
LD_LIBRARY_PATH 3-57
LIBRARY_MATCH 3-57
LIBPATH 3-57
listed alphabetically 3-10
listed by topic 3-69
manipulating in Windows environments 3-8
modifying settings 3-6
NODEFAC 3-58
ONCONFIG 3-58
OPT_GOAL 3-60
OPTCOMPIND 3-59
OPTMSG 3-59
OPTOFC 3-60
overriding a setting 3-5, 3-35
PATH 3-61
PDQ_PRIORITY 3-61
PLCONFIG 3-62
PLOAD_LO_PATH 3-63
PLOAD_SHMBASE 3-63
PSORT_DBTEMP 3-63
PSORT_NPROCS 3-64
RTREE_COST_ADJUST_VALUE 3-65
rules of precedence in UNIX 3-7
rules of precedence in Windows 3-10
scope of reference 3-8
SERVER_LOCALE 3-13
setting 3-8
at the command line 3-4
in a configuration file 3-4
in a login file 3-4
in a shell file 3-5
in Windows environments 3-8
with the System applet 3-8
setting in autoexec.bat 3-9
SHELL_PATH 3-65
standard UNIX system 3-3
STM_CACHE 3-66
TERM 3-66
TERMCAP 3-66
TERMINF 3-67
THREADLIB 3-67
TOBIGINT 3-68
types of 3-3
unsetting 3-6, 3-9, 3-35
UNELEMENTNAME 3-68
view current setting 3-6
where to set 3-5
XFER_CONFIG 3-68
Equality (=) operator 2-9
Era-based dates 3-35
Error message files 3-23
esql command 3-16, 3-47
ESQL/C
DATETIME routines 3-31
esql command 3-16
long identifiers 3-42
message chaining 3-59
multi-threaded applications 3-67
program compilation order 3-16
Exact numeric data types 1-58
Executable programs 3-61
Execute privilege 1-36, 3-58
Explicit cast 1-14, 2-44
Explicit pathnames 3-9, 3-27
Explicit temporary tables 3-29

X-8 IBM Informix Guide to SQL: Reference
Files (continued)
   temporary for SE  3-31
   termcap, terminfo  3-52, 3-66, 3-67
FILETOBLOB function  2-5
FILETOCLOB function  2-8
Filtering mode  1-34, 1-53
Finalization function  1-10
FIXED column format  1-26, 1-27
Fixed point decimal  2-13, 2-20, 3-24
Fixed-length UDT  1-56
FLOAT data type
   built-in casts  2-42
   defined  2-15
   display format  3-23, 3-24
Floating-point decimal  2-13, 2-15, 2-28, 3-23
Foreign key  A-6, B-2
Formatting
   DATE values with DBDATE  3-20
   DATE values with GL_DATE  3-33
   DATETIME values with DBTIME  3-31
   DATETIME values with GL_DATETIME  3-33
   DECIMAL(p) values with DBFLTMASK  3-23
   FLOAT values with DBFLTMASK  3-23
   MONEY values with DBMONEY  3-24
   SMALLFLOAT values with DBFLTMASK  3-23
Formatting mask
   with DBDATE  3-20
   with DBFLTMASK  3-23
   with DBMONEY  3-24
   with DBTIME  3-31
   with GL_DATE  3-33
   with GL_DATETIME  3-33
FRACTION keyword
   DATETIME qualifier  2-10
   INTERVAL qualifier  2-17
   FRAGMENT BY clause  3-29
Fragmentation
   distribution strategy  1-28
   expression  1-28, 3-19, 3-22
   list  1-28
   PDQ_PRIORITY environment variable  3-62
   PSORT_NPROC environment variable  3-65
   round robin  1-28
   setting priority levels for PDQ  3-61
   sysfragauth data  1-27
   sysfragments data  1-28
FROM keyword  1-7, 1-15
Function keys  3-52
Functional index  1-31, 2-40
Functions
   for BLOB columns  2-5
   for CLOB columns  2-8
   for MULTIBSET columns  2-21
   support for complex types  2-39
fwrtable gcc option  3-47
G
   gcc compiler  3-47
   Generalized-key index
      sysindexes data  1-30
      sysnewdepend data  1-33
      syssizepository data  1-41
   Generic B-trees  1-31
   geography table in sales_demo database  B-3
   GET DIAGNOSTICS statement  1-25
   getenv utility  3-4
   GL_COLLATE table  1-50
   GL_CTYPE table  1-50
   GL_DATE environment variable  2-9, 3-20, 3-21
   GL_DATETIME environment variable  2-12, 3-20
   Global Language Support (GLS)  3-25
   Global network buffer pool  3-42
   GLOBAL_DETACH_INFORM environment variable  3-36
   Globally detached index  1-29
   GLS environment variables  3-7
   GNU C compiler  3-47
   GRANT statement  1-42
   Graphic characters  3-52
   Greenwich Mean Time (GMT)  3-37
   GROUP BY clause  2-6, 2-29, 3-30
   Group informix  3-24
H
   Hash-join  3-39, 3-59
   Hashed columns  1-28
   Hasing parameters  1-46
   Heap size  3-56
   Hebrew locales  2-7
   Hexadecimal digits  3-22
   HIGH INTG keywords
      ALTER TABLE statement  2-33
      CREATE TABLE statement  2-33
   HIGH keyword
      PDQ_PRIORITY  3-61
      UPDATE STATISTICS  1-8, 1-24
   High-Performance Loader  3-26, 3-62
   Histogram  1-24
   Host language  1-59
   Host variable  2-5, 2-6, 2-29, 2-40
   HOUR keyword
      DATETIME qualifier  2-10
      INTERVAL qualifier  2-17
   HP-UX operating system  3-65
   HTML (Hypertext Markup Language)  2-8
   Hybrid fragmentation strategy  1-29
   Hyphen
      DATETIME delimiter  2-11
      INTERVAL delimiter  2-18
I
   I/O overhead  3-65
   IBM Informix Dynamic Server (IDS)  1-8
   IBM Informix ESQ/C  3-15, 3-21, 3-31, 3-42, 3-59
   IBM Informix Extended Parallel Server (XPS)  1-8, 3-10
   IBM Informix Storage Manager (ISM)  3-54, 3-55
   IBM_XPS_PARAMS environment variable  3-37
   IDS (Informix Dynamic Server)  1-8
   IFMX_CART_ALRM environment variable  3-38
   IFMX_OPT_FACT_TABS environment variable  3-38
   IFMX_OPT_NON_DIM_TABS environment variable  3-39
   IFX_DEF_TABLE_LOCKMODE environment variable  3-39
   IFX_DIRECTIVES environment variable  3-40
   IFX_EXTDIRECTIVES environment variable  1-23, 3-41
   IFX_LONGID environment variable  3-41
   IFX_NETBUF_PYTPool_SIZE environment variable  3-42
   IFX_NETBUF_SIZE environment variable  3-42
   IFX_NO_LRU_TUNING environment variable  3-43
   IFX_NO_TIMELIMIT_WARNING environment variable  3-43
   IFX_NODBPROC environment variable  3-43
   IFX_ONTAPE_FILE_PREFIX environment variable  3-44
Key
foreign A-6, B-2
generalized 1-33, 1-41
primary 1-21, 1-41, 1-53, A-5, B-5
Key scan 1-11
Key tables 3-49
Keyboard I/O
INFORMIXTERM setting 3-52
TERM setting 3-66
TERMCAP setting 3-66
TERMININFO setting 3-67
keytab file 3-49
Korn shell 3-5

L
Language
C 1-42, 3-16, 3-47
C++ 3-49
CLIENT_LOCALE setting 3-21
DBLANG setting 3-23
Extensible Markup Language (XML) 2-8
Hypertext Markup Language (HTML) 2-8
Informix ESQL/C 2-33, 2-40, 3-67
Java 3-14, 3-56
sql_languages information schema view 1-59
Stored Procedure Language (SPL) 2-39, 3-19, 3-22
syslangauth data 1-32
sysroutinelangs data 1-42
Large-object data type
defined 2-32
listed 2-31
LD_LIBRARY_PATH environment variable 3-57
Leaf pages 1-29
LIBERAL_MATCH environment variable 3-57
libos.a library 3-41
LIBPATH environment variable 3-57
LIKE keyword of SPL 2-25, 2-26
LIKE operator 2-6, 2-29, 2-46, 3-57
Linearized code 1-51
List
of data types 2-3
of environment variables 3-10
of environment variables, by topic 3-69
of system catalog tables 1-8
LIST data type, defined 2-18
LOAD statement 2-5, 2-6, 2-29, 3-22
Locales
collation order 1-50, 2-31
multibyte 2-7
of trace messages 1-50
right-to-left 2-7
specifying 3-69, 3-72
Lock-table overflow 3-40
LOCKMODE keyword 3-39
LOCOPY function 2-5, 2-8
LOG keyword
ALTER TABLE statement 2-33
CREATE TABLE statement 2-33
Logging mode 1-16
Long identifiers
client version 3-41
IFX_LONGID setting 3-41
Information Schema views 1-58
LOTOTFILE function 2-5, 2-8
LOW keyword
PDQPRIORITY 3-61
LOW keyword (continued)
UPDATE STATISTICS 1-24
Lowercase mode codes 1-39
Lowercase privilege codes 1-6, 1-16, 1-27, 1-47, 1-55
LVARCHAR data type
casting opaque types 2-44
coltype code 1-19
defined 2-19
M
Machine notes 3-53
Magnetic storage media 1-14
Mantissa precision 1-58, 2-13
manufact table in superstores_demo database B-10
Map file for C++ programs 3-49
MATCHES operator 2-6, 2-29, 2-31, 2-46, 3-57
MaxConnect 3-45, 3-46
MEDIUM keyword 1-8, 1-21, 1-24
Membership operator 2-46
Memory cache, for staging blobspace 3-50
Message file
specifying subdirectory with DBLANG 3-23
XBSA 3-54
Messages
chaining 3-59
error in sysservers 1-25
optimized transfers 3-59
reducing requests 3-60
trace message template 1-50
warning in sysservers 1-25
mi_collection_card() function 2-19, 2-21, 2-27
mi_db_error_raise() function 1-25
Microsoft C compiler 3-47
MINUTE keyword
DATETIME qualifier 2-10
INTERVAL qualifier 2-17
MISTRACE_OFF configuration parameter 1-50, 1-51
mkdir utility 3-24
MODERATE INTEG keywords
ALTER TABLE statement 2-33
CREATE TABLE statement 2-33
Modifiers
CLASS 1-39
COSTFUNC 1-39
HANDLESNULLS 1-39
INTERNAL 1-39
NEGATOR 1-39
NOT VARIANT 1-39
PARALLELIZABLE 1-39
SELCONST 1-39
STACK 1-39
VARIANT 1-39
MODIFY NEXT SIZE keywords 1-8
MONEY data type
built-in casts 2-42
defined 2-20
display format 3-24
international money formats 2-20
length (syscolumns) 1-20
MONTH keyword
DATETIME qualifier 2-10
INTERVAL qualifier 2-17
Multibyte characters
CLOB data type 2-9
VARCHAR data type 2-31
MULTISET data type
   constructor 2-39
   defined 2-21

N
   N setting
      sysroleauth.is_grantable 1-42
   Named ROW data type
      casting permitted 2-45
      defined 2-23
      defining 2-23
      equivalence 2-23
      inheritance 1-32, 2-23
      typed tables 2-23
   NCHAR data type
      collation order 2-21
      conversion to CHAR 3-26
      defined 2-21
      multibyte characters 2-21
   Negator functions 1-39
   Nested dot notation 2-40
   Nested-loop join 3-59
   Network buffers 3-42, 3-43
   Network environment variable, DBPATH 3-26
   NFS directory 3-30
   NO KEEP ACCESS TIME keywords
      ALTER TABLE statement 2-33
      CREATE TABLE statement 2-33
   no setting of NODEFDAC 3-58
   NODE configuration parameter 3-58
   NODEFDAC environment variable 3-58
   NOLOG keyword
      ALTER TABLE statement 2-33
      CREATE TABLE statement 2-33
   NONE setting
      ISM_ENCRYPTION 3-55
      JAVA_COMPILER 3-56
   Nonprintable characters
      CHAR data type 2-7
      TEXT data type 2-29
   VARCHAR data type 2-30
   NOT NULL constraint
      collection elements 2-19, 2-21, 2-27, 2-39
      syscoldepend data 1-17
      sysconstraints data 1-22
   NOT NULL keywords 2-6, 2-19, 2-29
   NOT operator 2-46
   NULL value
      allowed or not allowed 1-10, 1-19
      BOOLEAN literal 2-6
      BYTE data type 2-6
      TEXT data type 2-29
   Numeric data types
      casting between 2-42
      casting to character types 2-42
      listed 2-51
   NVARCHAR data type
      collation order 2-22
      conversion to VARCHAR 3-26
      defined 2-22
      length (syscolumns) 1-20
      multibyte characters 2-22

O
   Object mode of database objects 1-33
   Object-relational schema B-1
   ODBC driver 3-57, 3-65
   OFF setting
      IFX_DIRECTIVES 3-40, 3-41
      PDQPRIORITY 3-62
   ON setting
      IFX_DIRECTIVES 3-40, 3-41
      ON-Bar utility 3-55
      ONCONFIG environment variable 3-58
      onconfig.std file 3-58
      onconfig.xls file 3-58
      oninit command 3-40
   Online transaction processing (OLTP) 1-29
   onload utility 2-5, 2-6, 2-29
   opload utility 3-26, 3-63
   onstat utility 3-3, 3-37
   onutils utility 3-37
   Opaque data types
      cast matrix 2-45
      comparing 2-44
      defined 2-22
      smart large objects 2-32
      storage 2-19
      sysxtddesc data 1-54
      sysxtdtypes data 1-55
   OPCACHEMAX configuration parameter 3-50
   OPEN statement 3-60
   Operator class
      sysams data 1-11
      sysindices data 1-32
      sysopclasses data 1-34
      Operator precedence 2-45
      OPT_GOAL configuration parameter 3-60
      OPT_GOAL environment variable 3-60
      OPTCOMIND configuration parameter 3-59
      OPTCOMIND environment variable 3-59
   Optical cluster
      INFORMIXOPCACHE setting 3-50
      sysblobs.type 1-14
      sysopclstr data 1-35
   Optimizer
      setting IFX_DIRECTIVES 3-40
      setting IFX_EXTERNDIRECTIVES 3-41
      setting OPT_GOAL 3-60
      setting OPTCOMIND 3-59
      setting OPTOFCC 3-60
      sysdistrib data 1-24
   Optimizer directives
      sysdirectives data 1-23
      OPTMSG environment variable 3-59
      OPTTOFC environment variable 3-60
      OR operator 2-46
      ORDER BY clause 2-6, 2-29, 3-30
      orders table in superstores_demo database B-8, B-9, B-10
   Ordinal positions 2-18
   Output support function 2-19
   output() support function 2-39
   Overflow error 2-13
   Owner routines 1-39, 3-58

P
   PAGE lock mode 1-48, 3-40
   Parallel distributed queries, setting with PDQPRIORITY 3-61
Parallel sorting, setting with PSORT_NPROCS 3-63
Partial characters 2-7
PATH environment variable 3-61
Pathname
  for C compiler 3-47
  for C++ map file 3-49
  for client or shared libraries 3-57
  for concsm.cfg file 3-47
  for connectivity information 3-52
  for database server 3-26
  for dynamic-link libraries 3-57, 3-65
  for environment-configuration file 3-7
  for executable programs 3-61
  for installation 3-49
  for keytab file 3-50
  for message files 3-23
  for parallel sorting 3-63
  for remote shell 3-28
  for smart-large-object handles 3-63
  for temporary .jar files 3-56
  for termcap file 3-66
  for terminfo directory 3-67
  for XBSA messages 3-54
  for xfer_config file 3-68
  separator symbols 3-61
PDQ
  OPTCOMBIND environment variable 3-39
  PDQ/PRIORITY environment variable 3-61
  PDQ/PRIORITY configuration parameter 3-62
  Percentage (%) symbol 3-31
Period
  DATE delimiter 3-21
  DATETIME delimiter 2-11
  INTERVAL delimiter 2-18
Permissions 3-4, 3-24
PLCONFIG environment variable 3-62
plconfig file 3-62
PLOAD_LO_PATH environment variable 3-63
PLOAD_SHMBASE environment variable 3-63
PostScript 2-8
Precedence rules
  for casts 2-44
  for lock mode 3-40
  for SQL operators 2-45
  for UNIX environment variables 3-7
  for Windows environment variables 3-10
Precision
  of currency values 2-20
  of numbers 1-58, 2-13, 2-15, 2-16, 2-28
  of time values 2-10, 2-16, 2-34, 2-37
PREPARE statement 1-49
Prepared statement 1-49
Primary access method 1-11, 1-46
Primary key 1-22, 1-41, 1-53, 2-25, 2-26, A-1, B-5
Primary thread 3-52
printenv utility 3-6
Printing with DBPRINT 3-28
Private environment-configuration file 3-7, 3-36
Private network buffer pool 3-42, 3-43
Private synonym 1-48
Privilege
  default table privileges 3-58
  on columns (syscolauth table) 1-16
  on procedures and functions (sysprocauth table) 1-36
  on table fragments (sysfragauth table) 1-27
  on tables (sysstabauth table) 1-47
  on the database (sysusers table) 1-52
Privilege (continued)
  on UDTs and named row types (sysxtdtypeauth) 1-55
  product table in sales_demo database B-3
Protected routines 1-40
Protected rows 2-15
Pseudo-machine code (p-code) 1-36
PSORT_DBTEMP environment variable 3-63
PSORT_NPROCS environment variable 3-64
Public synonym 1-46, 1-48
public user name 1-57
Purpose functions 1-11
Push-down hash join
  dimension tables 3-39
  fact tables 3-38
putenv utility 3-4
Q
Qualifier field
  DATE 2-10
  EXTEND 2-37
  INTERVAL 2-16
  UNITS 2-37
Query optimizer
  defined 1-8
  directives 3-40, 3-41
  push-down hash-join plans 3-39
  sysprocplan data 1-40
Quoted string
  DATE and DATETIME literals 2-36
  DELIMIDENT setting 3-35
  INTERVAL literals 2-18
  invalid with BYTE 2-6
  invalid with TEXT 2-29
  LVARCHAR data type 2-19
R
R-tree index 3-34, 3-65
Read committed 1-60
Read uncommitted 1-60
recv() support function 2-39
References privilege 1-16, 1-47
region table in superstores_demo database B-10
Reject file 1-27
Relational operators 2-7, 2-46
Remote database server 3-15, 3-36
Remote shell 3-28
Remote tape devices 3-29
RENAME SEQUENCE statement 3-68
Repeatable read 3-59
Replica identifier 1-28
Resource contention 3-62
Resource Grant Manager (RGM) 1-29
Resource privilege 1-7, 1-52
REVOKE statement 1-47
Right-to-left locales 2-7
Role
  default role 1-52
  INFR_ROLE_SEP setting 3-53
  sysroleauth data 1-42
  sysusers data 1-52
  Role separation 3-53
  Round-robin fragmentation 1-28
Statements of SQL

ALTER OPTICAL CLUSTER 1-36
ALTER SEQUENCE 3-68
ALTER TABLE 1-8, 1-49, 3-68
CLOSE 3-60
CONNECT 3-26, 3-27, 3-48, 3-51
CREATE ACCESS METHOD 1-11
CREATE AGGREGATE 1-10
CREATE CAST 1-14, 2-43
CREATE DATABASE 3-26
CREATE DISTINCT TYPE 1-55, 2-15, B-13
CREATE EXTERNAL TABLE 1-26, 1-27
CREATE FUNCTION 1-42, 3-58
CREATE IMPlicit CAST B-13
CREATE INDEX 1-3, 1-30, 1-31, 1-33, 1-41, 1-49, 3-34
CREATE OPAQUE TYPE 1-55, 2-22
CREATE OPERATOR CLASS 1-34
CREATE OPTICAL CLUSTER 1-35, 1-36
CREATE PROCEDURE 1-36, 1-42
CREATE ROLE 1-42, 1-52

Index  X-15
stores_demo database (continued)
   manufa ct table columns A-4
   primary-foreign key relationships A-5
   stock table columns A-3
   structure of tables A-1
   strings option of gcc 3-47
Structured Query Language (SQL) 3-17
Subscripts 2-6, 2-29
SUBSTRING function 1-7
Subtable 1-28, 1-32, B-8, B-14
Subtype 1-32, 2-23
Summary
   of data types 2-3
   of environment variables, by topic 3-69
   of environment variables, by type of server 3-10
   of system catalog tables, by type of server 1-8
superstores_demo database
   call_type table columns B-5
   catalog table columns B-5
   cust_calls table columns B-6
   customer table columns B-6, B-7
   defined B-4
   items table columns B-8
   manufact table columns B-10
   orders table columns B-8, B-9, B-10
   primary-foreign key relationships B-15, B-17
   sales_rep table columns B-11
   stock table columns B-11
   stock_discount table columns B-12
   structure of tables B-5
Super table 1-32, B-8, B-14
Supertype 1-32, 2-23
Support functions
   DISTINCT data types 2-40
   OPAQUE data types 2-22, 2-39
   routine identifier 1-37
Symbol table 1-38
Synonym
   sys synonyms data 1-45
   syssyntable data 1-46
   systables data 1-48
   USETABLENAME setting 3-68
Syntax diagrams
   reading in a screen reader C-1
   sysaggregates system catalog table 1-10
   syssams system catalog table 1-11
   sysattribtypes system catalog table 1-13
   sysblbs system catalog table 1-13
   sysbuiltintypes table 1-3
   syscasts system catalog table 1-14, 2-41
   syschecks system catalog table 1-15
   syscheckuddep system catalog table 1-15
   sysscolattribs system catalog table 1-15
   sysscolastepend system catalog table 1-17
   sysscolastems system catalog table 1-18
   syssconstraints system catalog table 1-21
   syscrd database 1-3
   sysdbcloset
   disabling with IFX_NODBPROC 3-43
   sysdbclosel() routine 3-4
   sysdbopen
   disabling with IFX_NODBPROC 3-43
   sysdbopen() routine 3-4
   syslogdefaults system catalog table 1-22
   sysdepend system catalog table 1-23
   sysdirectives system catalog table 1-23
sysdistrib system catalog table 1-24
sysdomains system catalog table 1-25
syserrors system catalog table 1-25
sysextcols system catalog table 1-26
sysextendfiles system catalog table 1-26
sysexternal system catalog table 1-27
sysfragauth system catalog table 1-27
sysfragments system catalog table 1-28
sysindexes system catalog table 1-30
sysindexes system catalog tables 1-31
sysinherits system catalog table 1-32
syslangauth system catalog table 1-32
syslogmap system catalog table 1-33
sysmaster database 1-3
   initialization 3-3, 3-51
   versus system catalog tables 1-3
sysnewdepend system catalog table 1-33
sysobjstate system catalog table 1-33
sysopclasses system catalog table 1-34
sysopclstr system catalog table 1-35
sysproclauth system catalog table 1-36
sysprobody system catalog table 1-36
sysproccolumns system catalog table 1-37
sysprocedures system catalog table 1-37
sysprocplan system catalog table 1-40
sysreferenc es system catalog table 1-41
sysreposit ory system catalog table 1-41
sysroleauth system catalog table 1-42
sysroutinelangs system catalog table 1-42
sysseclabelauth system catalog table 1-45
sysseclabelcomponentsystem catalog table 1-43
sysseclabelcomponents system catalog table 1-42
sysseclabelnames system catalog table 1-45
sysseclabels system catalog table 1-44
syssecpolicies system catalog table 1-43
syssecpolicycomponents system catalog table 1-44
syssecpolicyexceptions system catalog table 1-44
syssequences system catalog table 1-45
syssequences system catalog tables 1-45
sys synonyms system catalog table 1-45
syssyntable system catalog table 1-46
syssystables system catalog table 1-46
syssystabauth system catalog table 1-46
syssystables system catalog table 1-48
System administrator (DBA) 1-3
System applet 3-8
System catalog tables
   access methods 1-11, 1-46
   accessing 1-7
   altering contents 1-8
   authorization identifiers 1-52
   casts 1-14
   columns 1-18
   complex data types 1-13, 1-55
   constraint violations 1-53
   constraints 1-15, 1-17, 1-21
   data distributions 1-24
   database tables 1-48
   default values 1-22
   defined 1-2
   dependencies 1-23, 1-33
   example
      syscolauth 1-5
      sysscolumns 1-4
      sysindexes 1-6
      systabauth 1-5
      systables 1-4
   external tables 1-26, 1-27
System catalog tables (continued)
  fragmentation 1-27, 1-28
  indexes 1-30, 1-31, 1-41
  inheritance 1-32
  list of tables 1-8
  messages 1-25, 1-50
  operator classes 1-34
  optical clusters 1-35
  privileges 1-16, 1-27, 1-47, 1-52, 1-55
  programming languages 1-32, 1-42
  referential constraints 1-21, 1-41, 1-53
  roles 1-42
  routines 1-36, 1-37, 1-40
  security label components 1-42
  sequence objects 1-45
  simple large objects 1-13
  smart large objects 1-15
  synonyms 1-45, 1-46
  trace classes 1-50
  trace messages 1-50
  triggers 1-51, 1-52
  updating 1-8
  use by database server 1-3
  user-defined aggregates 1-10
  User-Defined data types 1-54, 1-55
  views 1-48, 1-53
System catalogs
  sysaggregates 1-10
  sysams 1-11
  sysattrtypes 1-13
  sysblbs 1-13
  syscasts 1-14
  syschecks 1-15
  syscheckuddep 1-15
  syscolattribs 1-15
  syscollauth 1-16
  syscoldepend 1-17
  syscolumns 1-18
  sysconstraints 1-21
  sysdefaults 1-22
  sysdepend 1-23
  sysdiindirects 1-23
  syss decentral 1-24
  sysdomains 1-25
  syserrors 1-25
  sysextcoils 1-26
  sysextendiles 1-26
  sysexternal 1-27
  sysfragauth 1-27
  sysfragments 1-28
  sysindexes 1-30
  sysindexes 1-31
  sysinherits 1-32
  syslangauth 1-32
  syslogmap 1-33
  sysnewdepend 1-33
  sysobjstate 1-33
  sysopclasses 1-34
  sysopclsttr 1-35
  sysprocauth 1-36
  sysprobody 1-36
  sysproccolumns 1-37
  sysprocedures 1-37
  sysproclplan 1-40
  syssreferences 1-41
  sysrepository 1-41
  sysroleauth 1-42
System catalogs (continued)
  sysroutinelangs 1-42
  syseclabelauth 1-45
  syseclabelcomponents 1-43
  syseclabelcomponents 1-42
  syseclabelnames 1-45
  syseclabels 1-44
  sysecpolicies 1-43
  sysecpolicycomponents 1-44
  sysecpolicyexemptions 1-44
  sysequences 1-45
  syssynonyms 1-45
  sysssystemtable 1-46
  systabamdata 1-46
  systabauth 1-47
  systables 1-48
  systracelss 1-50
  systracemsgs 1-50
  systrigbody 1-51
  systriggers 1-51
  syssusers 1-52
  syssviews 1-53
  syssviolations 1-53
  syssxhdatasources 1-54
  syssxasourcetyp 1-54
  syssxtddesc 1-54
  syssxtddtypeauth 1-55
  syssxtddtypes 1-55
  SYSTEM() command, on NT 3-54
  systracelssystem catalog table 1-50
  systracemsgs system catalog table 1-50
  systrigbody system catalog table 1-51
  systriggers system catalog table 1-51
  syssusers system catalog table 1-52
  syssutil database 1-3
  syssuuid database 1-3
  syssviews system catalog table 1-53
  syssviolations system catalog table 1-53
  syssxhdatasources system catalog table 1-54
  syssxasourcetyp system catalog table 1-54
  syssxtddesc system catalog table 1-54
  syssxtddtypeauth system catalog table 1-55
  syssxtddtypes system catalog table 1-55, 2-22, 2-23

T
  tabid 1-4, 1-49
Table
  changing a column data type 2-41
  dependencies, in syssdepend 1-23
  fragmented 1-28
  hashing parameters 1-46
  hierarchy 1-28, 1-32, 2-23, B-14
  inheritance, syssinherits data 1-32
  lock mode 3-39
  nonfragmented 3-34
  separate from large object storage 2-32
  structure in superstores_demo database 2-5
  synonyms in syssynonyms 1-45
  system catalog tables 1-10, 1-55
  temporary 3-29, 3-30
  temporary in SE 3-31
  typed, and named ROW type 2-23
  untyped, and unnamed ROW 2-24
  Table-based fragmentation 1-28
  Table-level privileges
    PUBLIC 1-57

Index  X-17
Table-level privileges (continued)
  sysfragauth data 1-27
  systabauth data 1-5, 1-47
Tape management
  setting DBREMOTECMD 3-28
Temporary dbspace 3-29
Temporary files 3-30
  in SE, specifying directory with DBTEMP 3-31
  setting DBSPACETEMP 3-29
  setting PSORT_DBTEMP 3-63
Temporary tables 3-29
  in SE, specifying directory with DBTEMP 3-31
  specifying dbspace with DBSPACETEMP 3-29
TERM environment variable 3-66
TERMCAP environment variable 3-66
termcap file
  setting INFORMIXTERM 3-52
  setting TERMCAP 3-66
Terminal handling
  setting INFORMIXTERM 3-52
  setting TERM 3-66
  setting TERMCAP 3-66
  setting TERMINFO 3-67
terminfo directory 3-53, 3-67
TERMINFO environment variable 3-67
TEXT data type
  casting to CLOB 2-29
  collation 2-29
  defined 2-28
  increasing buffer size 3-17
  inserting values 2-29
  length (syscolumns) 1-21
  nonprintable characters 2-29
  queries 2-29
  restrictions
    in Boolean expression 2-29
    with GROUP BY 2-29
    with LIKE or MATCHES 2-29
    with ORDER BY 2-29
  setting buffer size 3-17
  sysblobs data 1-13
  sysfragments data 1-28
  with control characters 2-29
Text editor 3-22
Thousands separator 2-20
thread flag of ESQL/C 3-67
THREADLIB environment variable 3-67
Time data types
  arithmetic 2-34
  length (syscolumns) 1-20
  listed 2-31
time table in sales_demo database B-4
Time values
  DBCENTURY setting 3-18
  DBDATE setting 3-20
  DBTIME setting 3-31
  GL_DATETIME settings 3-33
  USEOSTIME parameter 2-12
Time zone, specifying 3-37
Time-limited licenses (IFX_NO_TIMELIMIT_WARNING) 3-43
TO keyword
  DATETIME qualifier 2-10
  EXTEND function 2-35
  INTERVAL qualifier 2-16
  TOBIGINT environment variable 3-68
  TODAY operator 1-22, 3-37
  Trace class 1-50
Trace statements 1-50
Trailing blank spaces 3-57
Transaction isolation level 1-60, 3-59
Transaction logging 1-16, 1-60, B-1
Triggers
  creation-time value 3-19, 3-22
  sysobjstate data 1-33
  systrigbody data 1-51
  systriggers data 1-51
TRUE setting
  BOOLEAN values 2-6
  CFIRST 3-16
  ISM_COMPRESSION 3-54
  ISM_ENCRYPTION 3-55
  sysams table 1-11
Truncation 2-7
TYPE keyword 2-24

U
UDT indexes 3-65
Unary arithmetic operators 2-46
Uncommitted read 1-60
Under privilege 1-47
Unique constraint 2-25, 2-26
Unique index 1-30, 2-25
Unique keys 1-11
Unique numeric values
  SERIAL data type 2-25
  SERIAL8 data type 2-26
UNITS operator 2-9, 2-34, 2-36, 2-46
units table in superstores_demo database B-12
UNIX
  BSD, default print utility 3-28
  environment variables 3-3
  PATH environment variable 3-61
  System V
    default print utility 3-28
    terminfo libraries 3-53, 3-67
    temporary files 3-64
    TERM environment variable 3-66
    TERMCAP environment variable 3-66
    TERMINFO environment variable 3-67
UNLOAD statement 3-17, 3-22
Unnamed ROW data type
  declaring 2-24
  defined 2-24
  inserting values 2-25
  unset utility 3-6
  unsetenv utility 3-6
Unsetting an environment variable 3-6
Untyped table 1-49
Update privilege 1-16, 1-28, 1-47, 3-58
UPDATE statements 1-53, 2-6, 2-29, 3-44
UPDATE STATISTICS FOR PROCEDURE statement 1-40
UPDATE STATISTICS statement 1-31, 3-33
  and DBUPSPACE environment variable 3-33
  effect on sysdistrib table 1-24
  syindices (index statistics) 1-36
  syindices data 1-32
  update system catalog 1-8
Update trigger 1-52
Uppercase mode codes 1-39
Uppercase privilege codes 1-6, 1-16, 1-27, 1-47, 1-55
Usage privilege 1-55
USEOSTIME configuration parameter 2-12
User environment variable 3-10
User informix 1-8, 1-14, 2-41
User name 1-60
User privileges
syscolauth data 1-16
sysfragauth data 1-27
syslangauth data 1-32
sysprocauth data 1-36
systabauth data 1-47
sysusers data 1-52
sysxtdtypeauth data 1-55
User-defined aggregates 1-10
User-defined casts 1-14, 2-43
User-defined data types
 casting 2-43
 casting into built-in type 2-41
 opaque 2-41
 sysxtdesc data 1-54
 sysxtdtypes data 1-55
User-defined routines
 casts (syscasts) 1-14
 check constraints (syscheckudrdep) 1-15
 error messages (syserrors) 1-25
 for opaque data types 2-22
 functional index 3-34
 language authorization (syslangauth) 1-32
 privileges 1-36, 3-58
 protected 1-39
 secondary access method 1-21
 sysprocedures data 1-37
USETABLENAME environment variable 3-68
Utilities
archecker 3-14
chkenv 3-5, 3-6
DB-Access 1-7, 1-57, 3-8, 3-17, 3-23, 3-50, B-1
dbexport 3-22
dbload 2-5, 2-6, 2-29
dbschema 1-26, 1-27, 1-40, 3-68
dce_login 3-50
env 3-6
export 3-5
gcc 3-47
getenv 3-4
ifx_getenv 3-8
ifx_putenv 3-8
imcadmin 3-45, 3-46
load 1-27
lp 3-28
lpr 3-28
MaxConnect 3-46
ON-Bar utility 3-54, 3-55
oninit 3-40
onload 2-5, 2-6, 2-29
onupload 3-26, 3-63
onstat utility 3-37
onutil 3-38
onutils 3-37
onxfer 3-68
printenv 3-6
putenv 3-4
set 3-9
setenv 3-6
Setnet32 3-8
source 3-5
unset 3-6
unsetenv 3-6, 3-35
vi 3-23

V
VARCHAR data type
collation 2-31
conversion to NVARCHAR 3-25, 3-26
declared 2-30
length (syscolumns) 1-20
multibyte characters 2-31
nonprintable characters 2-30
storing numeric values 2-31
Variable-length packets 3-44
Variable-length UDT 1-56
VARIANT routine 1-39
vi text editor 3-23
View
columns view 1-58
Information Schema 1-56
server_info view 1-59
sql_languages view 1-59
sysdepend data 1-23
sysindexes view 1-32
syssynonyms data 1-45
syssyntable data 1-46
systabauth data 1-47
systables data 1-48
sysviews data 1-53
tables view 1-58
Violations
sysobjstate data 1-34
sysviolations data 1-53
Virtual machine 3-14, 3-56
Virtual processors 3-65
Visual disabilities
 reading syntax diagrams C-1

W
Warning message 1-25, 3-17
WHERE keyword 1-7, 1-15, 2-29
Whitespace characters 3-57
Whitespace in identifiers 3-35
Window borders 3-52
Windows environments
 manipulating environment variables 3-8
 setting environment variables 3-8

X
X setting
 sysms.am_sptype 1-11
 systabauth.tabauth 1-47
X/Open
 compliance 1-59
 Information Schema views 1-57
 server_info view 1-59
XBSA
 debugging records 3-55
 message log file 3-54
 shared library 3-55
XFER_CONFIG environment variable 3-68
xfer_config file 3-68
XML (Extensible Markup Language) 2-8
XOR setting 3-55
XPG4 standard 1-58, 1-59
XPS (Extended Parallel Server) 1-8, 3-10, B-1
Y

Y setting

    DBDATE  3-20
    DBTIME  3-32
    sysroleauth.is_grantable  1-42

Year 2000  3-18

YEAR keyword

    DATETIME qualifier  2-10
    EXTEND function    2-35
    INTERVAL qualifier  2-17

Year values, two and four digit  2-12, 3-18, 3-20, 3-32

yes setting

    NODEFDAC  3-58

YES setting

    columns.is_nullable  1-58
    sql_languages.integrity  1-59

Z

Zero (0)

   C null as terminator  2-31
   DBDATE separator     3-21
   DECIMAL scale        2-13
   hexadecimal digit    3-22
   IFX_DIRECTIVES setting  3-40, 3-41
   IFX_LONGID setting    3-42
   IFX_NETBUF_PVTPOOL_SIZE setting  3-42
   INFORMIXOPCACHE setting  3-50
   integer scale        1-59, 2-13
   ISM_DEBUG_LEVEL setting  3-55
   OPTCOMPIND setting    3-59
   OPTMSG setting        3-59
   OPTOFC setting        3-60
   padding of 1-digit years  3-18
   padding with DBFILMASK  3-23
   padding with DBTIME    3-32
   PDQPRIORITY setting    3-61
   PSORT_NPROCS setting   3-65
   STMT_CACHE setting     3-66
   sysams values         1-11
   sysfragments.hybdpos   1-29
   sysindices.nrows       1-31
   systables.type_xid     1-49
   sysxdtypes values      1-56

   zip column  B-9
   zipcode column A-2, B-9